CAMPUS
DE EXCELENCIA
INTERNACIONAL

Universidad Politécnica de Madrid
E.T.S. de Ingenieros Informáticos

Automatic design of Ant Colony Optimization for Permutation Problems

Javier Pérez

Directores:
Prof. Thomas Stützle
Dr. Manuel López-Ibáñez

2013/2014

Abstract

In this study, we present a framework based on ant colony optimization (ACO) for tackling combinatorial problems. ACO algorithms have been applied to many different problems, focusing on algorithmic variants that obtain high-quality solutions. Usually, the implementations are re-done for various problem even if they maintain the same details of the ACO algorithm. However, our goal is to generate a sustainable framework for applications on permutation problems. We concentrate on understanding the behavior of pheromone trails and specific methods that can be combined. Eventually, we will propose an automatic offline configuration tool to build an effective algorithm.

Abstract

En este trabajo vamos a presentar un framework basado en la familia de algoritmos *ant colony optimization* (ACO), los cuales están diseñados para enfrentarse a problemas combinacionales. Los algoritmos ACO han sido aplicados a diversos problemas, centrándose los investigadores en diversas variantes que obtienen buenas soluciones. Normalmente, las implementaciones se tienen que rehacer, inclusos si se mantienen los mismos detalles para los algoritmos ACO. Sin embargo, nuestro objetivo es generar un framework sostenible para aplicaciones sobre problemas de permutaciones. Nos centraremos en comprender el comportamiento de la sendas de feromonas y ciertos métodos con los que pueden ser combinados. Finalmente, propondremos una herramienta para la configuraron automática *offline* para construir algoritmos eficientes.

Acknowledgements

I would like to thank Thomas Stützle and Manuel López-Ibáñez, not only for giving me the opportunity of developing this thesis, but also for their help and patience. I would also like to thank all people I met during the last months I spent at IRIDIA.

to ...

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Inspired by the behaviour of some ant species, ant colony algorithms are a paradigm that can solve discrete combinatorial problems. Ant colonies can be described as distributed systems that exploit the communication abilities of their members to tackle complex tasks that in many cases exceed the capabilities of a single member. Although a colony of ants may look simple; there are a lot of questions to be answered for understanding how ants can interact between them to reach particular aims.

Over last years, a large number of researchers have been developing mathematical models of this natural cooperative intelligence, facing a wide variety of real problems with different ant colony algorithms. From this knowledge, we would like to extract the common things that everyone would want to have in its own library. Statements that allow us to create new algorithms based on the work previously developed; either functions or procedures that perform abstract operations to obtain particular results.

This thesis is split into four chapters: the first section describes the outline of the biological analogy and show a real experiment with a determinate ant specie. It also explains the computational theory and give a brief overview of stochastic models based on ants' behaviour. The second part studies the literature about

this topic, focusing on representative ant algorithms and different problems to be tackled. This section comprises both the second and the third chapter of this thesis. Then, we propose the new framework and analyze the results for the different problems described ahead. Eventually, in the sixth chapter, we are going to exploit the automatic design to obtain the best possible configurations of the developed framework for solving well-known problems working under the software developed.

1.1 From real ants to artificial ants

The visual perception in many species of ants is only rudimentary, further more some species are totally blind. For that reason, most of the communication among individuals or between individuals and the environment is based on the use of chemical substances called pheromones. In particular, there is a specific type of pheromone, called trail pheromone, that some ant species deposit for creating a path on the ground. With the guide of this trail pheromone, ants can discover the route between the nest and the source of food.

With the purpose of investigating this behavior, Deneubourg and colleagues [3] suggested the double bridge experiment: connect a nest of ants and a source of food with two different paths to study the effects of the trail pheromone under supervised conditions. They ran experiments in which they modified the ratio of the length between the two paths. At the start, the ants were totally free to choose between the two paths to move towards the source of food and the percentage of the ants choices for one or the other paths was observed over time. Results showed that, in most of the experiments in which one of the two paths was longer than the other, the ants were capable of discovering the shortest path between the origin point and the destination point, that is, between the nest and the source of food.

This outcome could be explained as follow: in the initial phase of the experiment there is no trail pheromone in any path, for that reason ants could choose with the same probability either of the two paths. The same number of ants could

opt for a path or the other, but stochastic oscillations may occur favouring one over the other. Later, due to the fact that one path is shorter, ants that selected the minimal path will return before, reinforcing the trail pheromone for future selections, such that the trail pheromone begins to accumulate faster on the short path and biases the decision of the rest of the colony on its favour.

Similarly, ants deposit trail pheromone both on their forward paths and on their backward paths. It is reasonable that this is necessary to obtain convergence of the ant colony towards the shortest path. If we consider a model in which trail pheromone is deposited only during the forward step or the backward step, the result is that the ant colony is not capable to find the shortest path.

It is also interesting to see what happens when the ant colony discovers a new shorter connection between the nest and the source of food once it had already converged to another path. In this case, the new short path is just selected sporadically and the colony continues to use the previous shorter path. This can be explained by the high pheromone concentration and slow evaporation of pheromone on the shortest path. The lifetime of the pheromone was comparable to the duration of the experiment, which means that the evaporation was too slow to allow the ant colony to forget the non-optimal path.

1.2 Stochastic Model

Deneubourg and colleagues [3] proposed a simple stochastic model that describes the behaviour observed in the double bridge experiment. In the model, ψ ants per second pass through the bridge in both directions with a constant speed v, depositing one unit of trail pheromone on the corresponding path. Knowing the shortest (l_s) and the longest length (l_s) distance between the nest and the food; an ant that has chosen the short path will take $t_s = l_s/v$, while an ant that has chosen the other path will take $r * t_s$, where r is the proportion between paths. Hence, the probability that one ant selects the shortest path is given by the following formula:

$$p_{os}(t) = \frac{(t_s + \varphi_{os}(t))^\alpha}{(t_s + \varphi_{os}(t))^\alpha + (t_s + \varphi_{ol}(t))^\alpha} \tag{1.1}$$

This model assumes that the amount of trail pheromone on a path is equivalent to the number of ants that have used this path in the past (φ_{os} and φ_{ol}) ,and suggests α equals to 2 based on their experiments. No pheromone evaporation is taken into account by the model. So, the behaviour of the continuous model is given by a set of differential equations, describing the instantaneous variation of the trail pheromone on the short path or the long path at time t:

$$\begin{aligned} d\varphi_{os}/dt &= \psi p_{ds}(t - t_s) + \psi p_{os}(t) \\ d\varphi_{os}/dt &= \psi p_{dl}(t - r \cdot t_s) + \psi p_{ol}(t) \end{aligned} \tag{1.2}$$

In order to validate the stochastic model, in [3, 4] these equations were simulated using the Monte Carlo method: when the two paths have the same length, the ants converge toward one or the other with equal probability; when one path is longer, then ants choose the short path in the majority of the experiments.

1.3 Ant path cost

In the previous section, we have shown that is possible to reproduce the main behaviour of ant colonies with the set of differential equations of the continuous model of Deneubourg et al. [3]. However, our main goal is to define abstract algorithms that can be used to solve maximum and minimum cost problems on complete graphs, not only the simpler double bridge experiment.

With this goal in mind, let us consider a static graph $G = (N, A)$, where N is the set of $n = |N|$ nodes and A is the set of undirected arcs connecting them. The two points between which we want to establish a connexion are called source and destination. As typically done in the literate, we may consider to types of paths problems formulations:

(a) *The minimum path cost problem*: in which we try to minimize the sum of the arcs cost between the source and the destination.

(b) *The maximum path cost problem*: in which we try to maximize the sum of the arcs cost between the source and the destination.

Unfortunately, if we try to solve both the maximum cost problem and the minimum cost problem on the graph G using directly artificial ants, the following problem arises: the ants, who construct a solution, might produce loops. This result is a consequence of the forward trail pheromone update mechanism, when the first artificial ant deposits his pheromone, then biases the following decisions of the rest of the colony. Loops tent to become more attractive and ants can get stuck in them.

Even if this problem does not appear, the pheromone forward update mechanism causes that the shortest path is no longer preferred like in the double bridge experiment. Hence, since this problem is because of the forward pheromone, it may be easily solved: removing the mechanism and allowing to deposit trail pheromone only on backward updating. But, as we said before: the system does not work anymore, not even in the simplest case of the double bridge experiment.

We need to extend the capabilities of artificial ants in a way that, while maintaining the most important characteristics of real ants, they are able to solve both minimum cost path problem and maximum cost path problem on generic graphs. In particular, artificial ants are limited by their memory in which they can store the partial paths that they have followed so far, as well as the cost of the paths that they have already used. Improving the memory skills of ants enables the implementations of a longer number of useful behaviours that will allow them to

build solutions efficiently depending on the type of problem to be solved.

These original behaviours are the following:

1. Probabilistic solution construction biased by pheromone trails, avoiding forward trail pheromone update mechanism.
2. Determinist backward trail pheromone update mechanism, avoiding the creation of loops.
3. Evaluation of the quality of the created solution, determining the quantity of trail pheromone to deposit.

1.4 Time complexity

Many combinatorial problems are hard to solve and require significant resources. According to the theory of computation complexity, a wide variety of problems are well-known to belong to the NP-Hard class, in other words, the time needed to solve an instance of this sort of problem grows exponentially related to its instance size in the worst case.

The reason of this behavior is easy to explain: the double bridge experiment has just two paths to be explored but if we increase the number of paths, the feasible space of search becomes higher, having to figure out which is the best path between all possibilities. In fact, not only there exist one computational class, also we can define other classes based on the Turing Machines' model[1]. These classes are represented in the figure 1.1 and they are the following.

[1] A Turing Machine is basically a static state machine with a theoretically infinity memory represented in the form of an infinite tape. During each step, depending on the current state of the machine and on the symbol on the current cell of the tape, the machine writes a new symbol on the tape, changes the state and moves the tape right or left.

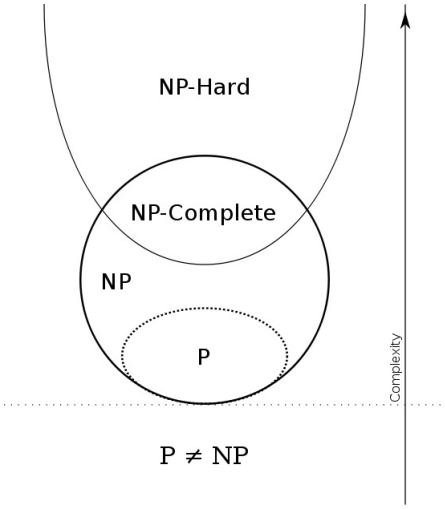


Figure 1.1: Euler diagram for P, NP, NP-Hard and NP-Complete classes [5]

Class *P*: a problem X belongs to P if X can be solved in polynomial time by a deterministic Turing Machine.

Class *NP*: a problem X belongs to NP if X can be verified in polynomial time by a deterministic Turing Machine.

Class *NP*-Hard: a problem X is NP-Hard when every problem in NP is reducible to X.

Class *NP*-Complete: a problem X is NP-Complete when it is both in NP and NP-Hard.

Additionally, the P and NP classes contain *decision problems*, that is, we only expect a boolean response either yes or no. For that reason, a problem in NP-Hard is at least as hard as whatever problem in NP without necessarily being a *decision problem*. Obviously, since a deterministic Turing Machine is a simplification of a nondeterministic Turing Machine, this implies that $P \subseteq NP$. However, the question whether $P = NP$ or $P \subset NP$ is still open.

Chapter 2

The Metaheuristic

In our days, it is commonly accepted the impact of computer science in our lives, when we speak about an algorithm o even a heuristic mode to attack a problem it is easy to find one definition which we can agree with. However, it is unclear when we refer to a metaheuristic. This chapter intends to clarify the concept of metaheuristic to give the lector a small introduction of this wide field.

2.1 What is a Metaheuristic?

Informally, we could define a metaheuristic as a framework which can contain multiple mechanisms to be applied to different combinatorial problems with a relatively small number of modifications to adapt them. This is particularly useful for understanding how the combinatorial problems react to alternative methods in reasonable time, although we would like to propose another point of view:

> A metaheuristic is a set of policies that can be used to define heuristic methods applicable to a large set of different problems. This could be seen as a generic heuristic method designed to produce a fundamental line base that biases the specific problem algorithms toward regions of the search space that contain high quality solutions.

2.2 The ACO Metaheuristic

Ant colony optimization (ACO) is a metaheuristic in which a colony of artificial ants interact for reaching high quality solutions on a large number of discrete combinatorial problems. As in its biological analogy, cooperation between the members of the colony is the clue for the design of ant colony optimization: individual agents that communicate with the environment, creating indirect links with other agents of the colony.

In a straightforward way, we can simulate this natural environment through abstract data structures that will keep the artificial pheromone trails and will allow access to an entire ant colony to them. Intuitively, an ant colony optimization algorithm could be described as the relationship of three procedures that are proposed in algorithm 1.

Algorithm 1 ACO-Metaheuristic

```
1: procedure
2:     Set parameters and initialize pheromone trails
3:     while stop condition do
4:         ConstructAntsSolutions
5:         UpdatePheromoneTrails
6:         DaemonsActions
7:     end while
8: end procedure
```

After the initialization of the parameters, there are some methods that deposit an initial quantity of trail pheromone on the global system, then a main loop is repeated until a stop condition is reached. This termination may be varied between different stop conditions: number of solutions constructed, time used for the experiment or even if the colony of has found the best possible solution in the search space.

Into the main loop, we can find the following statements:

ConstructAntSolutions: being the main driver of a colony of ants, concurrently and iteratively constructs candidate solutions, making use of trail pheromone and heuristic information. Through the path, an ant moves by applying stochastic local decision over the feasible neighborhood, that means, those paths that are not explored yet by a particular ant. In this sense, ants incrementally build solutions for the combinatorial problem, and either once an ant has terminate or while the solution is being built, ants evaluate the partial or complete solution that will be used in the *UpdatePheromoneTrails* procedure to decide the quantity of pheromone to deposit depending on the quality of this candidate solution.

UpdatePheromoneTrails: this procedure attempts to modify the measure of trail pheromone, either increasing or decreasing the actual value of pheromone trails. The deposit of new trail pheromone makes that the possibilities to choose a path becomes higher. Differently, the pheromone evaporation may help us to forget the worst candidate solution, that is, to eliminate a percentage of trail pheromone on a path, moreover it avoids the faster convergence of the algorithms towards non-optimal solution, favouring the exploration of new regions of the search space.

DaemonsActions: used to implement general methods in a centralized way, that is, procedures that either involve more than one agent of the colony or can not be performed by an unique ant. The ants' solution is not guaranteed to be the optimal solution respect to the local search space, for that reason, the solution created could be improved by local search methods. Also, we may collect information that could be used to decide whether is useful to deposit trail pheromone according to statistical methods, influencing the search process toward a determinate search space.

2.3 Evolution over time

In this section we are going to present those ant colony optimization algorithms that are similar to the first approach into this metaheuristic: ant system (AS) proposed by Dorigo et al. [4]. We do not consider the use of local search for improving the quality of the solution generated.

2.3.1 Ant System

Originally, three alternative versions were proposed by Dorigo et al. [4]: *ant density*, *ant quantity* and *ant cycle*. The two first algorithms were characterized by the deposit of pheromone directly after a move from one original point to the adjacent point, instead the third algorithm just could deposit pheromone when an ant had chosen the complete solution. Nowadays, we refer to ant system as the old *ant cycle* due to the fact that the other two versions shown a poor performance and they were abandoned.

Ant system is constituted by the two main phases mentioned above: the artificial ants iteratively build solutions that start over a deterministic first position. Each step, ant k applies a probabilistic choice rule to decide which path is the next to visit. In detail, the probability that an ant k selects the path between the points (i, j) is given by the equations (2.1).

$$p_{ij}^k = \frac{[\tau_{ij}]^\alpha [\eta_{ij}]^\beta}{\sum_{l \in N_i^k} [\tau_{il}]^\alpha [\eta_{il}]^\beta}, \quad if\ j \in N_i^k \tag{2.1}$$

where $\eta_{ij} = 1/f_{ij}$ is a heuristic value available at the beginning, α and β are two parameters which determinate the influence of the trail pheromone τ_{ij} and the heuristic information and N_i^k is the feasible neighborhood of an ant k at the point i, that is, the set of possible new path to be explored.

After all the ants have constructed their solutions, the pheromone trails are updated. In the first step, ant system decreases the trail pheromone that this simulates the pheromone evaporation in a real environment, using the equation (2.2).

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij} \tag{2.2}$$

where $0 < \rho \leq 1$ is the rate of pheromone evaporation. This is a parameter that enables the algorithms to allow the infinite storage of pheromone trails and admits to forget a bad decision previously taken. Thus, all ants deposit its particular measure of trail pheromone on the solution already created following the equation (2.3).

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^{m} \triangle\tau_{ij}^{k} \tag{2.3}$$

where $\triangle\tau_{ij}^{k}$ is the amount of trail pheromone that an ant k deposits in the paths of its solution, defined by the following formula:

$$\triangle\tau_{ij}^{k} = \begin{cases} 1/C^{k} & if(i,j) \in T^{k} \\ 0 & otherwise \end{cases} \tag{2.4}$$

where C^{k} is computed as the evaluation of the objective function of the solution T^{k} built by an ant k.

2.3.2 Elitist Ant System

A first improvement based on ant system. The main idea was the introduction of reinforcement between the connections of the best solution discovered since the start of the algorithm. This additional feedback could be seen as a daemon action of the ant colony optimization metaheuristic, adding a quantity e/C^{bs} to the paths using in the best solution, where e is a parameter that defines the weight of the best-so-far solution S^{bs} and C^{bs} is the length of the solution T^{bs} . Then, the new equation for the pheromone update is given by the formula (2.5).

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^{m} \triangle\tau_{ij}^{k} + e \triangle \tau_{ij}^{bs} \tag{2.5}$$

where $\triangle\tau_{ij}^{bs}$ is defined by the equation 2.4 as follow:

$$\triangle\tau_{ij}^{bs} = \begin{cases} 1/C^{bs} & if(i,j) \in T^{bs} \\ 0 & otherwise \end{cases} \tag{2.6}$$

Similar to ant system, the elitist ant system implement the pheromone evaporation mechanism in the same way as we describe on the equation (2.2)

2.3.3 Rank Ant System

Another improvement of ant system is the rank ant system. Depending on the rank of an ant k, it is possible to put some quantity of trail pheromone, being the best-so-far ant the one that deposits the largest amount of trail pheromone. The colony of ants is sorted by quality of its solutions and the weight of the trail pheromone is estimated based on the rank r of the ant.

In each iteration of the rank ant system, just the $(w-1)$ best ranked ants are allowed to deposit trail pheromone, therefore the best ranked ant is considered the best-so-for ant, having a higher weight in the pheromone update mechanism. Thus, the pheromone trails update is given by the equation (2.7), (2.7) and (2.9).

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{r=1}^{w}(w-r) \triangle \tau_{ij}^{r} + w \triangle \tau_{ij}^{bs} \tag{2.7}$$

$$\triangle\tau_{ij}^{r} = \begin{cases} 1/C^{r} & if(i,j) \in T^{r} \\ 0 & otherwise \end{cases} \tag{2.8}$$

$$\triangle\tau_{ij}^{bs} = \begin{cases} 1/C^{bs} & if(i,j) \in T^{bs} \\ 0 & otherwise \end{cases} \tag{2.9}$$

2.3.4 Max-Min Ant System

Unlike with the previous two improvements based on ant system, $MAX-MIN$ ant system introduces several modifications in the main procedures of the ant colony metaheuristic: this algorithm exploits the features of the best-so-far and iteration-best ant, that is, in the actual iteration can only deposit trail pheromone either the best solution found in the global experiment or the local best solution found in the current iteration. Despite this, the algorithm could get stucked in regions of the search space in early stages of the exploration. For this reason, the algorithm provides a balance mechanism that avoids to concentrate the trail pheromone in specific regions: now trail pheromone have lower bounds and upper bounds. Also, the initial pheromone trails are initialized to the upper bound with a small evaporation rate compared to the other algorithms mentioned above.

$MAX - MIN$ ant system was the first ant colony optimization algorithm that introduced the concept of the reinitialization of pheromone trails, that is, when the system is not capable to improve the previous solution that has been generated in a certain number of iterations, the pheromone trails are reinitialized with the default value of the actual best so far solution. Hence, the final pheromone update rule can be described as the equation (2.10).

$$\tau_{ij} \leftarrow \tau_{ij} + \triangle\tau_{ij}^{best} \tag{2.10}$$

In the previous formula, $\triangle\tau_{ij}^{best}$ is the ant allowed to put trail pheromone, this ant may be either the best-so-far (2.11) or the iteration-best (2.12).

$$\triangle\tau_{ij}^{bs} = \begin{cases} 1/C^{bs} & if(i,j) \in T^{bs} \\ 0 & otherwise \end{cases} \tag{2.11}$$

$$\triangle\tau_{ij}^{ib} = \begin{cases} 1/C^{ib} & if(i,j) \in T^{ib} \\ 0 & otherwise \end{cases} \tag{2.12}$$

Obviously, the frequency of selection has influence in the algorithm. Whereas when it is the iteration best ant which updates the pheromone trails, then the number of paths that could be explored is higher, being the search space no directly fixed. However, when the trail pheromone is updated with the best-so-far ant, the space of search focuses close to T^{bs}.

2.3.5 Ant Colony System

The ant colony optimization algorithms presented in the above sections have introduced important improvements that obtain the better solutions than the original ant system. However, the ant colony system differs from all the previous algorithms.

Such is the case that the colony of ants builds the solution based on a pseudorandom proportional rule. In other words, not only ants make the best possible moves suggested by the pheromone trails and the heuristic information, also they can improve the exploration of search space. This behavior is defined in the equation (2.13), and requires to use the parameter q_0.

$$\begin{cases} argmax_{l \in N_i^k} \ [\tau_{il}]^{\alpha}[\eta_{il}]^{\beta} & if \ q \leq q_0 \\ equation \ (2.1) & otherwise \end{cases} \tag{2.13}$$

This algorithm also uses the backward update, that is, the best-so-far ant is only allowed to add pheromone trials after each iteration following the formula (2.14)

$$\tau_{ij} \ \leftarrow \ (1-\rho)\tau_{ij} + \rho \bigtriangleup \tau_{ij}^{bs} \tag{2.14}$$

$$\tau_{ij} \ \leftarrow \ (1-\delta)\tau_0 + \delta\tau_0 \tag{2.15}$$

In addition to the backward pheromone updating rule, in ant colony system the ants can use the forward pheromone trial update immediately after they choose a path in the construction phase. This mechanism is represented by the equation (2.15).

Chapter 3

NP-Hard Problems

After have gained a good knowledge of the different ACO algorithms presented on the literature, we are going to take into account four representative problems to solve using this approach. This chapter provides a detailed description of these problems and a brief description of the instances that will be used in the following experiments.

3.1 Traveling Salesman Problem

Being the most famous combinatorial problem and widely studied, the traveling salesman problem (TSP) is a problem that aims at finding the shortest possible trip through a set of cities, visiting each city once before returning the starting city.

The TSP can be represented by a complete graph $G = (N, A)$, with N being a set of $n = |N|$ nodes, also called cities, and A being the set of arcs that fully connect the nodes. A single pair (i, j) of A is assigned a value d_{ij} which represents the distance between the cities i and j. Then, TSP is the problem of finding the minimum length hamiltonian circuit of the graph, where a hamiltonian circuit is a closed tour along the cities, visiting exactly each city once. A solution to an instance of the TSP can be represented as a closed permutation of the cities, that means, the absolute position of a city in a tour is not important at all, just the rel-

ative order is important. In other words, there are n permutations that construct the same solution.

We can also distinguish between symmetric TSP's, where the distance of the arcs is independent of the direction of traversing the arcs, that is, $d_{ij} = d_{ji}$ for each pair of cities, and asymmetric TSPs, also called ATSP, where at least there is a pair of nodes (i, j) for which $d_{ij} \neq d_{ji}$.

The instances that we are going to use are divided into two categories:

TSPLIB: this library is a set of symmetric instances for the traveling salesman problem. This instances were use as reference to validate the correctness of the algorithm.

Generic: these instances are also symmetric traveling salesman problems that we propose to use both to the training phase and the testing phase of the automatic design. They are split into five groups on different sizes: from 1000 till 3000 cities.

3.2 Quadratic Assignment Problem

The Quadratic Assignment Problem (QAP) is considered one of the hardest optimization problems. This problem attempts to assign a set of facilities to a set of locations given a distance between the locations and a flow between the facilities.

The QAP can be represented as two n x n matrix: $A[a_{ij}]$ and $B[b_{uv}]$, where a_{ij} is the distance between the localization i and j, and b_{uv} is the flow between the facilities u and v. Thus, the QAP is the problem of finding the arbitrary permutation ϕ that minimizes the relationship between the flows and the facilities represented by the formula (3.1).

$$f(\phi) = \sum_{i=1}^{n}\sum_{j=i}^{n} b_{ij}a_{\phi(i)\phi(j)} \tag{3.1}$$

In the literature, we can find different sets of instances, but we only use the following:

QAPLIB: this library is a set of symmetric and asymmetric instances of the Quadratic Assignment Problem. Almost all researchers validate the fitness of the algorithms with this kind of instances, so we use them just as reference.

Generic: these instances are a bit different, they were generated specifically to be use both in the training phase and in the testing phase of the automatic design. They are classified either structured or unstructured and split into three group by different size: 60, 80 and 100.

3.3 Flow Shop Problem

The permutation flow shop problem consists in scheduling some jobs with a given processing time on different machines, keeping the sequence of jobs on all machines identical. It is commonly supposed that machines are identical, that is, they need the same time to process the same job.

In this problem, the objective is to find a sequence of jobs that minimizes the completion time C_{max}, also called *makespan*. Like in the previous combinatorial problems, we can define two groups of instances:

Taillard: this library comprises a set of instances of permutation flow shop problem that can be use to verifiy the quality of the solution generated.

Generic: these instances are just to use in both the training phase and the testing phase of the automatic design.

3.4 Linear Ordering Problem

The linear ordering problem is a permutation ϕ of rows and columns to maximize the sum of the elements in the upper tingle as the equation (3.2) represented

$$f(\phi) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} c_{\phi(i)\phi(j)} \tag{3.2}$$

The set of instances in this case is limited just to one:

LOLIB: this library contain a varied set of different instances, it contains real world instances based on the flows of the input and output of different economies, and random instances generated for validation of the quality of an algorithm. For that reason, we are going to use the real instances to have a reference, then we divided the instances called xLOLIb into two subgroups to obtain two sets of instances for the phases of training and testing of the automatic design.

Chapter 4

Redesign and New Concepts

As we have seen, the permutation problems always have slight differences that do not permit to develop a unique application for all of them. Due to the complexity of this sort of obstacles, we propose a new point of view for the ant colony system family algorithms maintaining the characteristic of the metaheuristic and holding all of its flexibility.

4.1 Framework Schema

Ant colony optimization algorithms share a common general schema that embeds the main characteristics of the natural ants systems. Our case is not a exception, the framework still maintains this schema as shown in the algorithm 2.

Algorithm 2 fACO Schema

```
1: procedure
2:     Set parameters and initialize pheromone trails
3:     while stop condition do
4:         ConstructAntSolutions
5:         SortAntColony
6:         UpdatePheromoneTrails
7:     end while
8: end procedure
```

Figure 4.1: Graphical representation of auxiliary permutation (O) and solution constructed (P). On the left, we represent an example that contains the successor representation with a non random distribution. On the right, we represent the other approach that contains the position representation with a random distribution.

4.2 Improved artificial ants

Commonly in the ant colony optimization paradigm, the artificial ants are capable to create a permutation using trail pheromone as global communication system, avoiding the forward update mechanism and evaluating the quality of the solution created. However, this straightforward representation creates a limited vision over the capabilities of the artificial ants. Our framework aims at improving the original artificial ants, giving them a new perspective of construction.

The algorithm builds solutions for a permutation problem using an auxiliary permutation (in some cases a partial permutation) that can be generated using preprocessing procedures, this will be explained in section 4.3.2.1. This auxiliary component allows to remember certain perturbations in order to constraint the search space or to offer guidance in the construction process. Therefore, the improved ants have two complementary behaviors added to the above described artificial ants.

1. Capability to remember previous permutations, making possible either to limit or explore the space of search.

2. Deterministic selection of positions in the permutation that have been constructed, being guided by partial solutions.

4.3 Dynamic adaptation

The main schema of this framework is a generalization based on the common methods used by all ant colony optimization algorithms. However, the new conceptual algorithm itself is not enough for allow switching between different combinatorial problems. In order to abstract the different problem characteristics, this framework proposes a dynamic approach that will deal with this problematic.

Parameter	Values
Representation	*successor*, *position*
Decision rule	*best*, *neighbour best*, *neighbour random*, *random*
First decision	*first random*, *first dummy*, *all random*
Iterative decision	*former*, *none*
Pheromone evaporation	*sysmmetric*, *asymmetric*, *neighbour*
Pheromone update	*best*, *alternative*, *schedule*, *interpolation*, *none*
Pheromone upgrade	*sysmmetric*, *asymmetric*, *neighbour*, *sumation*
Local search	*specific*, *two opt*, *generic*

Table 4.1: Dynamic feasible values of configuration for fACO

The proposed framework should adapt itself dynamically regarding the problem to be solved. On one hand it should be able to tackle different permutation problems using the proposed representation. On the other hand, it should be able to choose the best ant colony optimization approach for solving the permutation problem and to combine the different features of these algorithms to obtain a new and better performing algorithm.

The configuration of the framework is given by the parameters shown in table (4.1), each parameter defines the type of component used for each search space, these will be explained in the following of this chapter.

4.3.1 Initial phase

The framework has three manners to initialize its structure.

Static: the framework does not have any information to calculate the initial pheromone trails suggested by Dorigo and Stützle [4]. For that reason, we propose a method that initializes the pheromone trails to a large value for all relationship between two nodes.

Estimated: in this case, a initial value of the objective function is given as a parameter to the framework, thus it is applied one of the equation defined on Dorigo and Stüztle [4] to obtain the correct initial pheromone trails on the system.

Biased: the framework has been given a value for the objective function of the initial solution and also the permutation. Consequently it uses this information to deposit trail pheromone, and after it evaporates the pheromone trails following the equations in Dorigo and Stüztle [4].

Parameter	Values
Default Ant	*yes: 1 or no: 0*
Bound Lower	A *x* (A, B, C), *see annex A*
Bound Upper	$\{A\|B\}$ *x* (A, B, C, D, E), *see annex A*

Table 4.2: Parameters to initialize the framework

As the framework has a wide variety of variables that the internal state needs to set properly, the user needs to choose an appropriate combination of methods among the procedures developed, including the initialization of pheromone trail. Table 4.2 describes the parameters related to the described initialization processes, the parameter *default ant* shows if the information provided includes the permutation whereas the bound parameters select the equations to calculate the initial trail pheromone.

Once the initial pheromone trails are deposited, the framework looks for establishing the correct configuration of the internal methods for a determinate problem.

4.3.2 Generation of solutions

After having the initial pheromone trails, the following phase creates the permutations according to the size of the colony. Each ant can build one solution that is biased by the feedback given by the other members of the colony. Unlike classical construction phases illustrated in the literature, our schema (algorithm 3) provides an abstraction of main ant colony optimization algorithms and includes new methods to adapt the search to the permutation problem on execution time.

Algorithm 3 Construction Schema

```
1: procedure
2:     for all PopulationSize do
3:         RestartAnt
4:         ConditionalPreOrder
5:     end for
6:     repeat
7:         for all PopulationSize do
8:             if DecisionAvailable then
9:                 ConditionalPreOrder
10:                DecisionRule
11:                SaveAndRebuild
12:            end if
13:            if AntColonySystem then
14:                LocalUpdatePheromoneTrails
15:            end if
16:        end for
17:    until PermutationSize
18:    for all PopulationSize do
19:        ObjetiveFunction
20:        LocalSearch
21:    end for
22: end procedure
```

The auxiliary permutation manages the entire construction phase, that is, depend-

ing on the method selected we could construct different kind of solutions. Hence, we are capable to specify the representation of the permutation using values for parameter representation shown in the table (4.1).

4.3.2.1 Perturbation

In order to be able to construct different types of solutions, we need a mechanism that allow us to select the meaning of the actual permutation. There exist at least two representations for a single permutation: when the algorithm is capable to construct such permutation as the election of different paths of the search space, the representation is called *by successor*, and when the algorithms try to create a relationship between a resource and its receptor, is called *by position*. Depending on the type of permutation, the construction phase may vary.

By successor: This representation takes advantage of both mechanisms presented in the construction phase: *ConditionalPreOrder* and *ConditionalInOrder*. The first mechanism provides the method to initialize the permutation that an ant is going to construct. The second mechanism allow us to modify the internal order according to the initialization mechanism previously selected. Table (4.3) summarizes the parameters of this configuration: the *first decision* is equal to *ConditionalPreOrder* and the *iterative decision* is equal to *ConditionalInOrder*. In detail, designed specifically for the traveling salesman problem is proposed a *first random* decision and for the permutation flow shop problem is proposed a *first dummy* selection.

Parameter	Values
Representation	*successor*
First decision	*first random* or *first dummy*
Iterative decision	*former*

Table 4.3: Parameters to set successor representation

By position: Instead of using two mechanism in the construction phase, this representation just requires one mechanism to perform *ConditionalPre-Order*. In the initialization of the auxiliary permutation of the ants, a random sequence is generated and stored in the specific ant. Each member of the colony has a different initial sequence that provides all the information needed to build an entire permutation. Table (4.4) shows the parameters of this specific configuration:

Parameter	Values
Representation	*position*
First decision	*all random*
Iterative decision	*none*

Table 4.4: Parameters to set position representation

The framework has three dependent parameter to establish the correct relation between the representation of a permutation and the construction phase. This modifies the original behaviour of the colony and enables new features: ants are capable to remember permutations that they generate a priori, either permutations of an old iteration the colony or a random generated one.

4.3.2.2 Destruction

The destruction mechanism is a random method that rebuilds a solution previously created. The colony of ants is capable to generate a new solution based on the path discovered previously by another ant. This approach causes that ants obtain a new behavior to decide if the current position in the permutation has already been selected. This method is described in the algorithm 4: copying the *best-so-far* solution found, removing some previous decisions as the parameter *destruction size* indicates, and finally rebuilding the ant to proceed with the normal construction phase.

Algorithm 4 Destruction Schema

```
1: procedure
2:     ant ← bestSoFarAnt
3:     while destruction size do
4:         RemoveOldDecision
5:     end while
6:     RebuildBehaviours
7: end procedure
```

The values of the parameters to configure this mechanism are presented in table (4.5).

Parameter	Values
Representation	*successor or position*
First decision	*destroy*
Iterative decision	*none*
Destroy Size	$n \in range\ (1 \ldots Dimension)$

Table 4.5: Parameters to set the destruction mechanism

4.3.2.3 Local Update

Inherited from ant colony system, the local update is the mechanism that allow us to apply the forward pheromone trails updating. The effect of this kind of update is given by the previous formula presented in (2.15), being the needed configuration as simple as establish the family of algorithms as ant colony system instead of ant system as the table (4.6) shows.

Parameter	Values
Family	*ant system or ant colony system*
xi	$xi \in\ (0.00, 1.00)$

Table 4.6: Parameters to set inherited algorithms

4.3.2.4 Local Search

One of the technicalities that had to be solved here is how the optimization may be defined. Stützle and Hoos [15] shown that ant colony optimization algorithms are capable to construct good solutions only with the pheromone trail mechanism, but in many cases the algorithms do not find the best solutions. A proposal for improving the behaviors is to apply a generic optimization method that could explore the space of search. However, a new problem arises: in normal circumstances optimization methods are problem specific, they need to know a lot of information about an instance to avoid wasting a lot of time looking at the search space. In fact, the real problem is the size of the space of search related to the instance size, that is, the space of search becomes exponentially large.

Algorithm 5 Generic Local Search

```
1: procedure
2:     repeat
3:         improvement ← false
4:         for i ← 1 to dimension do
5:             for j ← 1 to dimension do
6:                 CheckMove(i, j)
7:                 if move improves then
8:                     ApplyMove(i, j)
9:                     improvement ← true
10:                end if
11:            end for
12:        end for
13:    until improvement
14: end procedure
```

In order to obtain a reasonable time of execution, we recommend to implement a problem specific optimization technique, but the framework also has a general optimization method that provides a first approach to the optimization of combinatorial problems. This method could be described as the basic movement of the local search, being presented in the algorithms 5 and using the following movements:

Transpose: Given a permutation ϕ, a transpose neighborhood movement is such that there are two consecutive member $i, j \in \phi$ that satisfy the condition of $position(i) = position(j) - 1$. Thus, the $i = \phi(j)$ and $j = \phi(i)$.

Exchange: Given a permutation ϕ, a exchange neighborhood movement is a extension of the previous movement such that there are two member $i, j \in \phi$. Thus, the $i = \phi(j)$ and $j = \phi(i)$

Insert: Given a permutation ϕ, a insert neighborhood movement is such that there are two member $i, j \in \phi$. Thus, the $\phi(i)$ is directly inserted after the position j.

Two opt: Given a permutation ϕ, a two opt movement creates a new permutation Φ through eliminating two links of the permutation ϕ, that is, having the two pairs of arcs $(\phi(i), \phi(x))$ and $(\phi(j), \phi(y))$, the modification creates new links, forming the arcs $(\phi(i), \phi(j))$ and $(\phi(x), \phi(y))$, and eliminates the above arcs.

4.3.3 Pheromone trails

When permutations have already being constructed for each member of the colony, the framework begins to deposit trail pheromone in the internal data structures. This information is the key of the correct working of ant colony optimization algorithms.

4.3.3.1 Evaporation

Although the pheromone evaporation is the simplest modification of pheromone trails, we define three different ways to implement this mechanism: symmetric, asymmetric and neighborhood. Each procedure depends on the problem attacked.

Symmetric: the trail pheromone between (i, j) and (j, i) is evaporated with the exact amount of pheromone ratio, that means, the direction between two positions in the permutation does not matter.

Asymmetric: opposite to the symmetric evaporation, the direction between two positions is relevant, then the ratio may be different.

Neighborhood: the framework uses a candidate set, therefore only the positions allocated in the candidate set are subject to evaporation.

4.3.3.2 Global Update

This mechanism encompasses two phases: the pheromone update and the pheromone upgrade. The values of their respective parameters are present in table (4.1). The values for the pheromone upgrade parameter are the same as in the pheromone evaporation parameter, adding a fourth method that represents the pheromone summation rule. This rule is a modification of the probabilistic formula presented in [2, 9] that allows to set priorities for the construction of the permutation. The alternative calculation is given by the formula (4.1)

$$p_{ij}^{k} = \frac{[\sum_{k=1}^{i} \tau_{kj}]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{l \in N_i^k} ([\sum_{k=1}^{i} \tau_{kl}]^{\alpha}[\eta_{il}]^{\beta})}, \quad if\ j \in N_i^k \tag{4.1}$$

The global update of pheromone trails have totally been modified and divided into two specific phases: in the first (algorithm 6), the framework uses the information that the colony has constructed in the phase of generate solutions, and secondly (algorithm 8) we use different indicators to make some statistics. Between the two phases, the framework implements a new mechanism called individual update (algorithm 7), that means, depending on the parameters given by the user, the framework could deposit an extra feedback.

Algorithm 6 Pheromone Trails Upgrade I

```
1: procedure
2:     GenericColonyUpdate()
3:     if extra feedback then
4:         ApplyPheromoneIndividualUpdate()
5:     end if
6:     StatatisticOfColony()
7: end procedure
```

Algorithm 7 Generic Colony Update

```
1: procedure
2:     weight ← weight general
3:     for i ← 1 to rank do
4:         ApplyPheromoneGeneralUpdate(i)
5:         weight ← weight - updateFactor
6:     end for
7: end procedure
```

This update is not a straightforward adaptation of the original ant colony optimization algorithms, it just a simulation of its behavior.

In contrast, the statistical phase checks simply two conditions that can be enabled automatically depending of the pheromone individual update. Eventually, the internal data structures are upgraded with the new information generated.

Algorithm 8 Pheromone Trails Upgrade II

```
1: procedure
2:     if limits enabled then
3:         CheckLimits()
4:     end if
5:     if restart then
6:         RestartPheromoneTrails()
7:     end if
8:     UpgradeInternalDataStructures()
9: end procedure
```

4.3.3.3 Individual

As we commented in the previous section, there exist a new concept called pheromone individual that consists in provide extra feedback to the colony with some special features. In general, this method is recognized as the update with the *best-so-far* ant of the colony, or with the *iteration-best* ant of the current iteration of the colony. However, depending of the ant colony optimization algorithm, this extra feedback could depend on a deterministic range as in rank ant system or could be just an ant with singular weight as in the elitist ant system.

In that sense, the default methods proposed for the pheromone individual update are the following:

Best: Directly the *best-so-far-ant* deposits pheromone trail according to an elitist weight. This method correspond to the one in elitist ant system where the elitist weight equals to parameter e.

Alternative Best: Similar to the above method, the *best-so-far* or the iteration-best applies pheromone according to an elitist weight, but also depending on a random variable that selects one or the other solution.

Alternative Static: Introducing the restart component, the pheromone individual may be updated with *best-so-far*, the *iteration-best* or the *restart-best*. This variation is the implementation version presented by [15] for algorithms without local search. The information needed for this formula is given by the parameter *schedule value*.

Alternative Adaptive: this mechanism includes the restart component and the *schedule value*, represents the implementation version presented by [15] for algorithms with local search. However, the framework calculates different levels to deposit pheromone trails based on the initial parameter *schedule value*. The level could be seen as an array, that is composed of four

different values given by the equation:

$$\begin{cases} \text{level 1: } \textit{schedule value} \text{ x } 1 \\ \text{level 2: } \textit{schedule value} \text{ x } 3 \\ \text{level 3: } \textit{schedule value} \text{ x } 5 \\ \text{level 4: } \textit{schedule value} \text{ x } 10 \end{cases} \tag{4.2}$$

Alternative Aggressive: Similarly to alternative adaptive, this method just is a modification of the level of the alternative adaptive method, making the space of search closed.

Interpolation: Instead of choosing between the *best-so-far*, the *iteration-best* or the *restart-best*, we use a merge update that is dynamically calculated at real time according to the convergence factor as proposed by Blum et al [2].

Moreover, It would be desirable to have a technique that allow us to create a new method to define extra feedback. In fact, this aspect is already present in the previous section, even so it will be described in more detail in the next chapters with a real experiment.

4.4 User interface

For linking the problem specific and the generic section of the code, we developed an interface that provides an entry to communicate some essentials characteristics, including the objective function that calculates how is the quality of the permutation created. The interface is divided in different files: each file is considered an individual unit, we propose the following schema:

AutoConfig: Containing a specify method, the main purpose is to create a default configuration that has been tested and defined as a generic well-know configuration for specific algorithms.

Heuristic: Depending on the problems, there are some problems that require a heuristic for deciding the best transitions between two paths. These heuristic should be programed in this file.

Interface: Being the principal unit, this describes the behavior of the objective function, the initialization and termination of the problem and the handler for the optimization.

Optimization: Implementing different approaches, this unit is dedicated to store the optimization methods to apply to the future solutions created by the ant colony optimization algorithms.

Parameters: Allowing the user to create new variables this is not a unit itself.

ReadProblem: Providing the statements to read the instances for each specific problem, it is an inheritance of the old algorithm to be backwards compatible.

Not all these units are obligatory, in most of the cases are only optional. However, if we want to approach a problem in the best way, it might be needed to have some methods that can provide customized alternative.

4.4.1 Mandatory units

Although it could be seen as a complex implementation, there are just two mandatory statements to start to work with the framework:

- Objective Function: belongs to the Interface unit, provides a value that represent the quality of the permutation.
- Read Problem Function: belongs the ReadProblem unit, provides the methodology for read an instance of a problem.

4.4.2 Optional units

As part of the interface, there are some methods to improve the final behavior.

- Heuristic Functions: belongs to the Heuristic unit, provides the heuristic values for the probabilistic elections.
- Candidate Set Procedure: belongs to the Interface unit, provides the methodology that establishs the candidates set for each possible allocation.
- Optimization Function: belongs to the Optimization unit, this function are the specific local search that the user can develop. If there are more than one option, a specific launcher will be needed.
- Default Value: belongs to the Interface unit, provides the entry for either the initial permutation or the initial best solution.
- Initialization and Termination Procedure: belongs to the Interface unit, this function should initialize the parameter that the problem specific code may need and release all the resources reserved
- AutoConfig Settings: belongs to the AutoConfig unit, provide a well-know configuration by default.

Chapter 5

Experimental Analysis

The goal of this chapter is the analysis of the framework in comparison with the well-know ACO algorithms presented in the literature. The experiments made have been proposed for understanding the behaviour of the framework behind different possible situations over the space of configurations and have been created to obtain a reference about the performance.

5.1 Overview

The framework is capable to deal with different combinatorial problems, approaching the solution through a set of ant colony optimization algorithms. We could apply novel algorithms that merge the features of the algorithms presented on the chapter 2, but we are going to focus just in simple configurations that allow us to compare the quality of the solutions obtained with other ant colony optimization algorithms presented in the literature. At this point, among the problems selected, only the traveling salesman problem gives a good reference due to its wide applications in the literature, in particular the ACOTSP code [14].

All the following experiments were executed under the same conditions, they were launched into the IRIDIA cluster on a single core AMD Opteron 6272 running at 2.1 Ghz with 16 MB of cache under Cluster Rocks Linux version 6.0/CentOS 6.3.

5.2 Dealing with the framework

As we have presented in the previous chapter, there are a large amount of parameters to configure. This task might be hard for a non expert user. Therefore, we propose guidelines that help the configuration of this framework.

5.2.1 Problem representation

In this particular case, we are referring to the traveling salesman problem, thus the features used to solve this problem are the followings¡:

1. Let the instance size be n, then the permutation size is $n + 1$.
2. Use of *nearest neighborhood heuristic* as initialization.
3. Generation of candidate set based of the distances between cities.
4. Construction according to successor.

Based on the table (4.1), the specification for the dynamic plugin is shown in the table (5.1).

Parameter	Values
Representation	*successor*
Decision rule	*neighbour random*
First decision	*first random*
Iterative decision	*former*
Pheromone evaporation	*neighbour*
Pheromone upgrade	*neighbour*
Local search	*specific*

Table 5.1: fACO parameters for TSP

5.2.2 ACO representation

Whilst the representation of the problems is unique for all the ant colony optimization algorithms, the configuration of those algorithms are individual (table 5.2 to 5.6).

Parameter	Values
Bound Lower	*none*
Bound Upper	*AD*
Default Ant	*0*
Destroy Size	*0*
Family	*ant system*
Limits	*no*
Rank	*25*
Schedule type	*none*
Schedule value	*n/a*
Update factor	*0.0*
Weight elitist	*0.0*
Weight general	*1.0*

Table 5.2: Ant system parameters for TSP simulated by fACO

The ACOTSP code developed by Stützle [14] has N parameters that must be set to define its behavior. These parameters are $\alpha = 1$ and $\beta = 2$ for the probabilistic function. The algorithm uses a candidate set, its size is set up to 20, the population size 25 and 10 for the ant system family and the ant colony system family respectively. The statistical parameters branching factor and lambda are configured to 1.01 and 0.05. Furthermore, the probabilistic decision parameter in the ant colony system p_0 is 0.98 and the deposition rate xi is 0.10. Finally, the parameter ρ has different values depending on the configuration as the reference software does.

Parameter	Values
Bound Lower	*none*
Bound Upper	*AD*
Default Ant	*0*
Destroy Size	*0*
Family	*ant system*
Limits	*no*
Rank	*25*
Schedule type	*best*
Schedule value	*n/a*
Update factor	*0.0*
Weight elitist	*25.0*
Weight general	*1.0*

Table 5.3: Elitist ant system parameters for TSP simulated by fACO

Parameter	Values
Bound Lower	*none*
Bound Upper	*AD*
Default Ant	*0*
Destroy Size	*0*
Family	*ant system*
Limits	*no*
Rank	*5*
Schedule type	*best*
Schedule value	*n/a*
Update factor	*1.0*
Weight elitist	*5.0*
Weight general	*6.0*

Table 5.4: Rank ant system parameters for TSP simulated by fACO

Parameter	Values
Bound Lower	*AB*
Bound Upper	*AD*
Default Ant	*0*
Destroy Size	*0*
Family	*ant system*
Limits	*yes*
Rank	*0*
Schedule type	*alternative adaptative*
Schedule value	*25*
Update factor	*0.0*
Weight elitist	*1.0*
Weight general	*0.0*

Table 5.5: Max-Min ant system parameters for TSP simulated by fACO

Parameter	Values
Bound Lower	*none*
Bound Upper	*AE*
Defualt Ant	*0*
Destroy Size	*0*
Family	*ant colony system*
Limits	*no*
Rank	*0*
Schedule type	*best*
Schedule value	*n/a*
Update factor	*0.0*
Weight elitist	*1.0*
Weight general	*0.0*

Table 5.6: Ant colony system parameters for TSP simulated by fACO

5.3 Performance obtained

In the early experiments, the framework was tested with the TSPLIB instances, but we did not appreciate significant variations between the solutions generated in ACOTSP and fACO. For that reason, we thought that hard instances may shown more differences (see annex B for complete results in detail). In bigger instances ACOTSP seems to be performing a bit better as shown the figure (5.1).

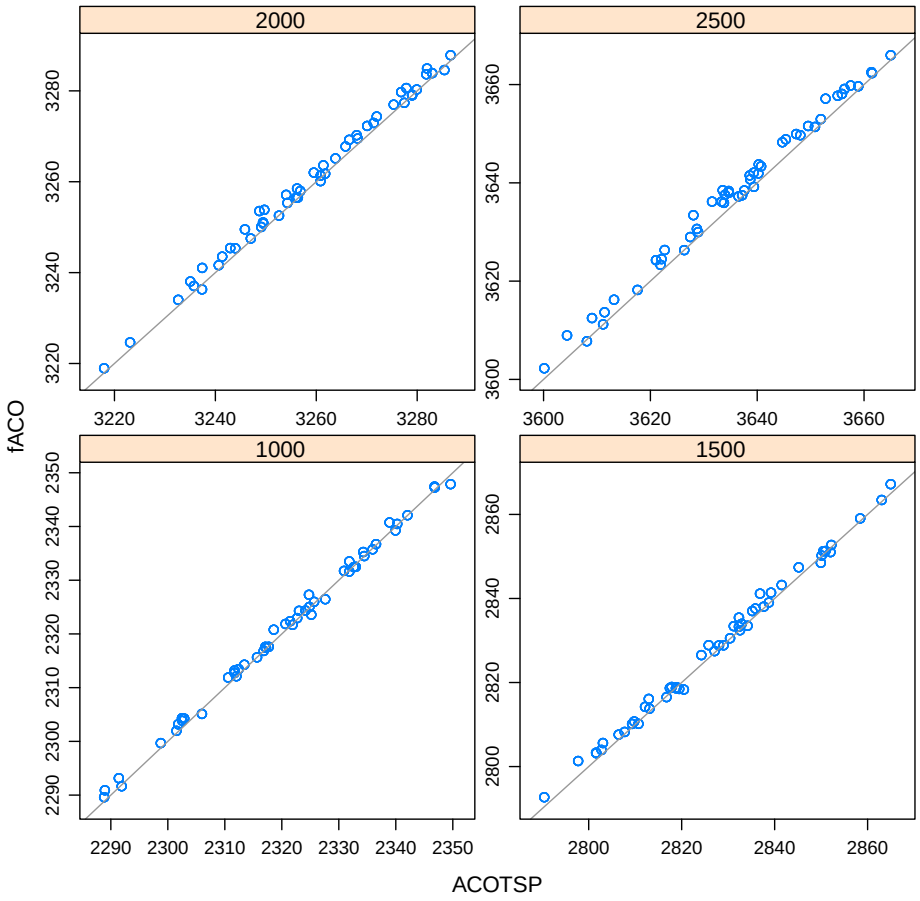


Figure 5.1: MMAS simulation with default parameters proposed by ACOTSP. The graphic compares the median of ten independent experiment for each instance. The evaluations are limited to 25000 without any specific termination time.

Chapter 6

Automatic Design

In the previous chapters, we presented a new ACO framework with several numbers of parameters to tackle different permutation problems. There are many options that allow us to set a grand amount of configurations to tune the framework in the most effective manner. For that reason, the remaining of this chapter gives a complete description about the automatic design and discusses the results that we obtained after tuning the parameters for each problem presented in the chapter 3.

6.1 What is iRace?

The irace tool [6] implements an iterated racing procedure. It is developed as an R package and it is built over the race package by Biratari (2003). The main purpose of irace is to automatically configure optimization algorithms, finding the most appropriate configuration given a space of configurations and limited by a set of tuning instances of an optimization problem.

Irace is an offline automatic configurator. Initially, irace generates a random set of configurations that are evaluated and compared over a set of instances of a problem. The main goal is to obtain an algorithm configuration that fits over the set of instances with the best probability, in each iteration the generated configurations are compared using a racing procedure in which the configurations that perform

statistically worst are eliminated of the race. In the next iterations, irace generates new configurations based on the previous ones.

6.2 Tuning Method

Automatic configuration allow us switching between different algorithms configurations and checking its quality in a reasonable time. For reaching this goal, we need to provide a generic software to the irace tool and a of instances of an optimization problem.

6.2.1 Overview

We have developed a flexible framework of ant colony optimization algorithms, different scenarios can be tackled resulting in a wide variety of cases of study. On a first phase, we propose to tune this framework to search an effective configuration based on the algorithms presented in chapter 2. Additionally, in the second phase we propose to merge the features of each ant colony optimization algorithm to obtain novel approaches. Obviously, the space of possible configurations is enormous, that is, our framework is capable to generate unrecognizable ant colony optimization algorithms such that the final performance may be equal but the procedure is hard to categorize.

Our methodology is defined as offline tuning, therefore the original set of instances is divided into two independent subsets: the training set and the testing set. The tuning is launched with a budget of 2.000 evaluations, and with the objective of minimizing the percentage deviation of the best known solution. All the tunings are performed under the same conditions, they were ran in the IRIDIA cluster on a multiple core AMD Opteron 6272 running at 2.1 Ghz with 16 MB of cache under Cluster Rocks Linux version 6.0/CentOS 6.3. For the analysis of the previous chapter, we assume that for each optimization problem the configuration proposed is the reference of performance and propose two independent tuning for each optimization problem.

6.2.2 Training Phase

Recapitulating, the list of parameter that are presented in our framework is given by the table (6.1). This table is divided into two main sections: the inherited parameters from the ACOTSP code and the parameters proposed for our framework, where the type indicates the type of the parameters (r: real, i: integer, c: categorial) and values indicates the range or set of possible values for the parameters.

As we presented in chapter 2, the probabilistic selection is biased by two parameters: α and β. This parameters establish the influence of the pheromone trails and the heuristic information. However, in certain cases, the heuristic information is unnecessary to construct solutions, that is, the use of heuristic information either does not improve the construction phase or does not exist for an optimization problem. In this cases, the parameter β is removed from the irace setup. Similarly, not only the traveling salesman problem has candidate sets, also this procedure could appear for other optimization problem. Our framework maintains two parameters from ant colony algorithms: *candidate set size* and *candidate set ls*, as their name suggest, they are the size for the candidate list into the construction phase and the local search phase respectively. In addition, two complementary parameters are created for generalize the concept: *candidate set type* and *candidate set window*. The first parameter proposed indicates the type of initialization that we have described in the chapter 4, the second parameter proposed sets a maximum number of elements available inside the current candidate set. Although the branching factor and the lambda correction are not consider as parameters in the original code, due to its implication in the internal statistics, our framework has the possibility to set an interval to probe the correct value. Also, we are going to run different configurations of tuning with a specific local search and the *dlb* parameter sets to 1. Following the list of parameters, we found two parameters specifically for when the *family* parameter is sets to *t*he *ant colony system*: p_0 and *xi*. The first parameter is the probability of choosing between the *best-so-far* solution or the actual candidate set inside the construction phase, whereas the second parameter is the influence of this selection into the local pheromone trails update.

parameter	*type*	*values*
ACO setting		
α	r	$[0.00, 5.00]$
β	r	$[0.00, 10.00]$
candidate set size	i	[2, 100]
candidate set ls	i	[2, 100]
dlb	c	(0, 1)
factor brach	r	[1.0, 1.3]
lambda	r	[0.05, 0.08]
local search	c	*specific*, *two opt*, *generic*, *none*
p_0	r	[0.00, 1.00]
population size	i	[2, 100]
ρ	r	[0.00, 1.00]
xi	r	[0.00, 1.00]
framework setting		
boud lower	c	$A\ x\ (A, B, C)$
bound upper	c	$\{A\|B\}\ x\ (A, B, C, D, E)$
candidate set type	c	(0, 1, 2)
candidate set windows	c	[2, 100]
default ant	c	(0, 1)
elected ants	i	[0, 1000]
elitist ants	i	[0, 1000]
family	c	*ant colony*, *ant colony system*
limits	c	(0, 1)
rank	i	[2, 100]
schedule type	c	*best*, *alternative*, *schedule*, *none*
schedule value	i	[0, 50]
update factor	c	(0, 1)
weight elitist	i	[0, 100]
weight general	i	[0, 100]

Table 6.1: Parameter list of fACO

The *population size* is the number of artificial ants that constitute the whole colony. The *rank* parameter gives the number of members of the ant colony that are allowed to deposit pheromone trail in a global manner, and through the *update factor* parameter we are able to change in runtime the weight of an ant of this colony. This deposition has a specific weight given by the parameter *weight general*. Alternatively, when a singular method of deposition of pheromone trail is needed, the *schedule type* parameter provides the mechanism that allow us switching between the different representations of the ant colony algorithms. In such case, the weight of the solution of an ant is given by the parameter *weight elitist* and whether this individual method needs an auxiliary value can be configured with the *schedule value* parameter.

The last issue about the parameters is the initialization of the pheromone trails in our framework. For this task, we have two parameter: *bound lower* and *bound upper*. The *limits* parameter actives the special mechanism that can limit the values of the pheromone trails. The *bound upper* always has to be configured because this method also represents the formula to calculate the initial pheromone trail. There are two complementary parameters that can provide a feedback over the initialization phase: *elected ants* and *elicits ants* (see annex A). In addition, we can propose an initial solution to avoid the initial calculation of the pheromone trails by the parameter *default ant*.

Finally, the representation of the different optimization problems follow the guidelines given in the chapter 3.

6.2.3 Testing Phase

After executing the irace tool, we obtained a wide variety of configurations. However, our goal is to check how the framework fits with the ant colony optimization algorithms. For that reason, we just want to validate the best candidate generated in each training phase and compare its performance with the reference assumed. The following sections discuss our result in details.

6.3 The generic algorithm

This section presents the five familiar ant colony algorithms that we have focused along this study compared with the setting reached with the automatic configuration procedure. For each optimization problem we show two configurations found as best candidates in two executions of irace.

Traveling Salesman Problem

The two configurations in the table (6.2) describe the top configuration reached.

parameter	*values (S1)*	*values (S2)*
ACO setting		
α	3.71	2.51
β	2.99	0.83
candidate set size	78	73
candidate set ls	28	15
factor branch	n/a	1.16
lambda	na	0.07
population size	72	91
ρ	0.90	0.96
framework setting		
bound lower	*none*	*AC*
bound upper	*AE*	*AA*
candidate set windows	34	36
default ant	0	0
limits	0	1
rank	12	n/a
schedule value	n/a	37

Table 6.2: Values of the tuned parameters configuration of fACO for TSP (I)

The first configuration (S1) corresponds to a rank ant system algorithm, whereas the second configuration defines a MAX-MIN ant system algorithm. Both configurations present a high value in the parameter ρ, that is, the evaporation of the pheromone trail is high compared to the default setting for the ACOTSP code. Moreover, candidate sets and population size look pretty similar, and the pheromone trails are more relevant than the heuristic as we expected.

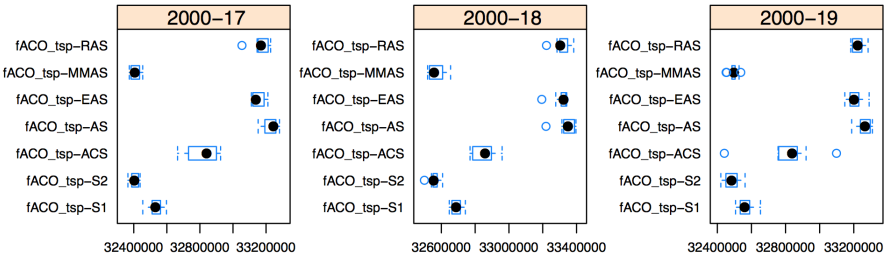


Figure 6.1: Results for the objective function for standard and autoconfig over the TSP. The graphic compares the median of ten independent experiment for each instance. The evaluations are limited to 25000 iterations and without any specific time.

According to the result shown in the figure (6.1), both candidates reached a good quality solution. In particular, the configuration (S2) achieved the best quality of the solutions among the rest of algorithms (see annex C for complete results in detail). However, the question that we would like to respond is how good is the solution compared to an automatic design on the ACOTSP code figure (6.2).

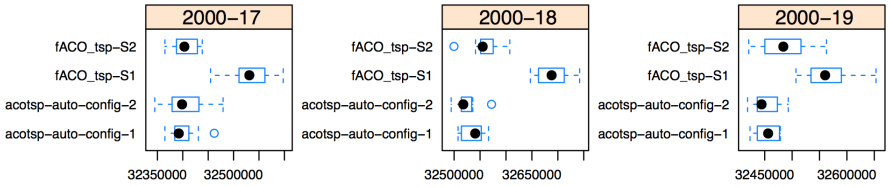


Figure 6.2: Results for the objective function on autoconfig ACOTSP and fACO for the TSP. The graphic compares the median of ten independent experiment for each instance. The evaluations are limited to 25000 without any specific termination time.

As the figure (6.2) shows, taking into account identical scenarios, the configuration of the simulate MAX-MIN ant system is quite similar. In certain cases, the ACOTSP is better than fACO, sometimes fACO is better than fACO (see annex C for complete results in detail). This alternative behavior is due to the stochastic process of the metaheuristic.

Quadratic Assignment Problem

When we proposed the representation of the this problem in table (6.3), the main differences were the candidate sets and the heuristic information. In other words, our algorithms do not need this information for construct a solution.

Even though, we have tried to obtain the best performance only with pheromone trails. Figure (6.3) shows the result reached by the framework (see annex C for complete results in detail), as in previous studies, the quadratic assignment problem has a high convergence in the first steps of construction, so the algorithms proposed by irace were two MAX-MIN ant system. The algorithms present a high value of the factor of branching that permits restart the pheromone trails in an easy way. Notice that it may be feasible to achieve better solutions with aggressive schedule, but in this step we are only using the default features of the ant colony optimization algorithms.

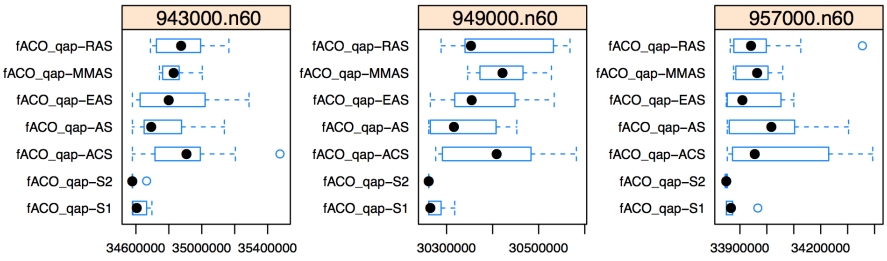


Figure 6.3: Results for the objective function for standard and autoconfig over the QAP. The graphic compares the median of ten independent experiment for each instance. The evaluations are limited to 25000 without any specific termination time.

parameter	*vaules (S1)*	*vaules (S2)*
ACO setting		
α	0.54	0.99
β	n/a	n/a
factor branch	1.18	1.24
lambda	0.08	0.07
population size	3	6
ρ	0.62	0.22
framework setting		
bound lower	*AA*	*AB*
bound upper	*AC*	*AE*
default ant	0	0
elected ants	n/a	53
elitist ants	669	n/a
limits	1	1
schedule value	35	33

Table 6.3: Values of the tuned parameters configuration of fACO for QAP (I)

Permutation Flow Shop Problem

Another problem that does not requires heuristic is the permutation flow shop problem. However, in this problem we propose a specific candidate sets that consists always in the best solution found. Both algorithms generated by irace are MAX-MIN ant system with a high level of evaporation as shown table (6.4). Moreover, the candidate sets suggests small windows with a large number of candidates.

Figure (6.4) presents our results (see annex C for complete results in detail). The quality of the solution is much better than the default configuration of the framework, perhaps due to the proximity to the heuristic solution.

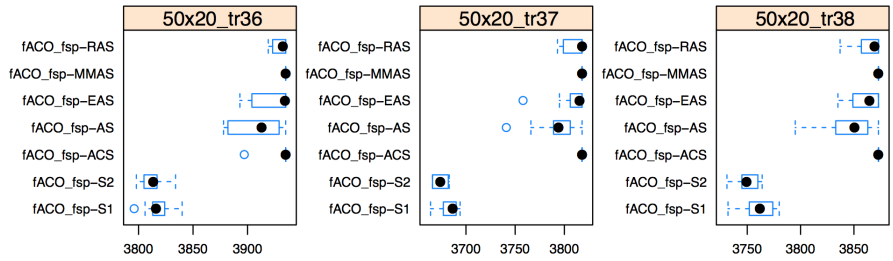


Figure 6.4: Results for the objective function for standard and autoconfig over the FSP. The graphic compares the median of ten independent experiment for each instance. The evaluations are limited to 25000 without any specific termination time.

parameter	*values (S1)*	*values (S2)*
ACO setting		
α	0.84	0.61
β	n/a	n/a
candidate set size	88	85
candidate set ls	18	48
factor branch	1.10	1.09
lambda	0.05	0.06
population size	100	40
ρ	0.98	0.87
framework setting		
boud lower	*AB*	*AC*
bound upper	*AA*	*AC*
candidate set windows	12	2
default ant	1	1
elected ants	316	n/a
elitist ants	n/a	457
limits	1	1
schedule value	50	24

Table 6.4: Values of the tuned parameters configuration of fACO for FSP (I)

Linear Ordering Problem

parameter	*vaules (S1)*	*vaules (S2)*
ACO setting		
α	3.60	1.19
β	1.34	0.55
candidate set size	100	70
candidate set ls	86	31
population size	5	7
ρ	0.38	0.40
framework setting		
boud lower	*none*	*AC*
bound upper	*BE*	*BB*
candidate set windows	10	9
default ant	0	0
limits	0	0
weight elitits	479	671

Table 6.5: Values of the tuned parameters configuration of fACO for LOP (I)

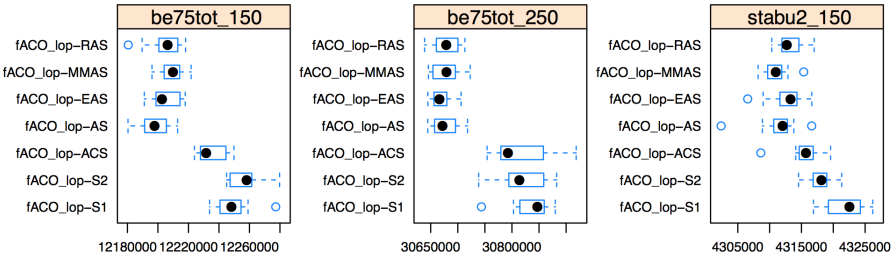


Figure 6.5: Results for the objective function for standard and autoconfig over the LOP. The graphic compares the median of ten independent experiment for each instance. The evaluation are limited to 25000 without any specific termination time.

Differently of the previous optimization problems, the linear ordering problem is a maximization problem that has not been previously applied to ant colony optimization. With this proposal, we want to probe the utility of the framework.

In the first step, we check the quality of a supposed default configuration without previous reference. The result were that the framework with this setting are far from good quality of solutions. Secondly, there are notable improvements when we apply the irace tool (see annex C for complete results in detail).

6.4 The free configuration

Once we have obtained a good configuration based on the ant colony optimization algorithms, the main issue is what happen when changing the configuration, letting irace to propose the best configuration without an established pattern. For this approach, we propose the configuration presented in the table (6.1) for all optimization problems, only that we are going to limit the space of configurations to *ant system family* and *schedule* configurations of individual pheromone trail update. Notice that the representation of the optimization problems does not change from the previous tuning.

Traveling Salesman Problem

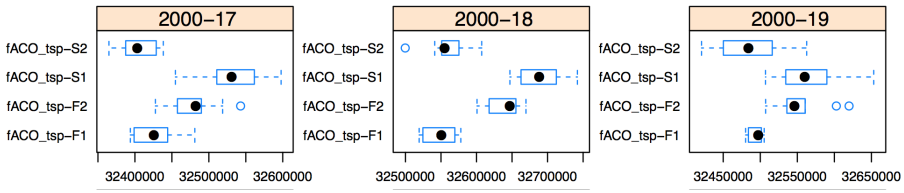


Figure 6.6: Results for the objective function on autoconfigs over the TSP. The graphic compares the median of ten independent experiment for each instance. The evaluation are limited to 25000 without any specific termination time..

Results shown in the Figure (6.6) indicate that the space of configurations proposed

is not good enough (see annex C for complete results in detail). The automatic design is worse than a default pattern. Perhaps, increasing the budget of irace could make feasible to reach a better solutions. Similarly, the trail pheromone is more important than the heuristic, maintaining a large set of candidates.

parameter	*vaules (F1)*	*vaules (F2)*
ACO setting		
α	4.69	3.22
β	5.99	1.91
candidate set size	80	80
candidate set ls	17	25
factor branch	1.23	1.03
lambda	0.08	0.05
population size	42	70
ρ	0.78	0.99
framework setting		
bound lower	*AC*	*AB*
bound upper	*AC*	*AB*
candidate set windows	10	14
default ant	0	1
elected ants	n/a	230
elitist ants	961	713
limits	1	1
rank	16	17
schedule value	39	45
update factor	1	0
weight elitist	460	296
weight general	16	39

Table 6.6: Values of the tuned parameters configuration of fACO for TSP (II)

Quadratic Assignment Problem

In the same way that the previous experiments, the final results (figure 6.7) do not show a significant improvement. This algorithms (table 6.7) have a low level of evaporation, but in overall they archive the same result as the automatic design based on the previous algorithms (see annex C for complete results in detail).

parameter	*vaules (F1)*	*vaules (F2)*
ACO setting		
α	3.17	3.01
β	n/a	n/a
factor branch	1.10	1.04
lambda	0.08	0.05
population size	49	70
ρ	0.04	0.16
framework setting		
bound lower	AB	AC
bound upper	AA	AB
candidate set windows	10	14
default ant	0	0
elected ants	319	n/a
elitist ants	n/a	762
limits	1	1
rank	47	33
schedule value	30	24
update factor	0	1
weight elitist	480	559
weight general	1	1

Table 6.7: Values of the tuned parameters configuration of fACO for QAP (II)

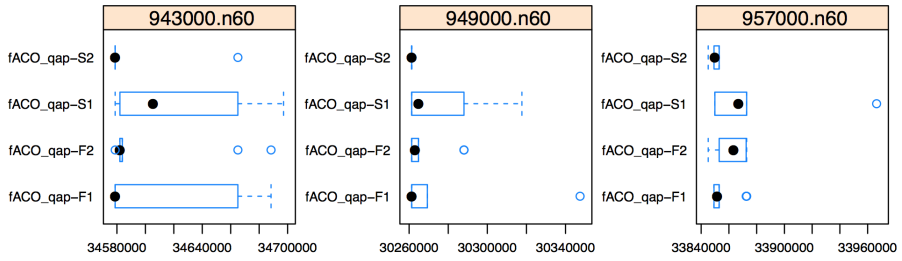


Figure 6.7: Results for the objective function on autoconfigs over the QAP. The graphic compares the median of ten independent experiment for each instance. The evaluations are limited to 25000 without any specific termination time.

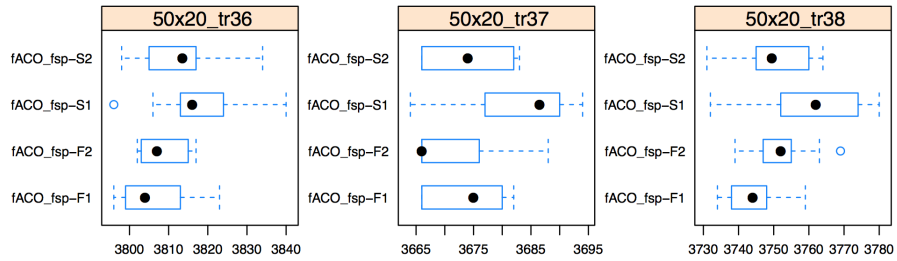


Figure 6.8: Results for the objective function on autoconfigs over the FSP. The graphic compares the median of ten independent experiment for each instance. The evaluations are limited to 25000 without any specific termination time.

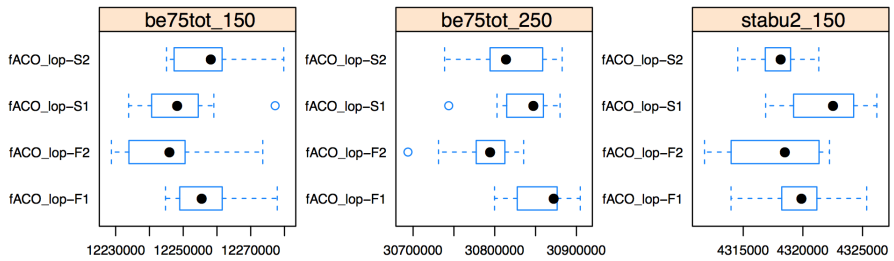


Figure 6.9: Results for the objective function on autoconfigs over the LOP. The graphic compares the median of ten independent experiment for each instance. The evaluations are limited to 25000 without any specific termination time..

Permutation Flow Shop Problem

The results of this experiment are the best among the free configuration. The framework was able to obtain a better solution compared to the previous reached (see annex C for complete results in detail), being the configuration very similar. Then, we believe that it is complex to achieve a better solution with our budget.

parameter	*vaules (F1)*	*vaules (F2)*
ACO setting		
α	3.11	4.24
β	n/a	n/a
candidate set size	99	100
candidate set ls	94	10
factor branch	1.28	1.25
lambda	0.06	0.06
population size	88	81
ρ	0.70	0.45
framework setting		
boud lower	*AA*	*AA*
bound upper	*AE*	*AD*
candidate set windows	13	98
default ant	1	1
limits	1	1
rank	60	63
schedule value	21	41
update factor	0	0
weight elitist	371	579
weight general	22	86

Table 6.8: Values of the tuned parameters configuration of fACO for FSP (II)

Linear Ordering Problem

In our last experiment we can observe that the population of ants is totally different. Instead of a small number of ants, the framework needs a large number of ants to achieve the same result (see annex C for complete results in detail).

parameter	*values (F1)*	*values (F2)*
ACO setting		
α	3.59	3.35
β	0.47	0.64
candidate set size	90	66
candidate set ls	68	14
factor branch	1.23	1.30
lambda	0.07	0.06
population size	66	87
ρ	0.81	0.50
framework setting		
boud lower	AC	AB
bound upper	BB	BD
candidate set windows	9	6
default ant	1	1
elected ants	n/a	246
limits	1	1
rank	54	58
schedule value	11	20
update factor	1	1
weight elitist	397	301
weight general	11	1

Table 6.9: Values of the tuned parameters configuration of fACO for LOP (II)

Chapter 7

Conclusions

The aim of thesis work was the implementation of a framework for optimization problems based on the ant colony optimization algorithms. In oder to reach meaningful modifications, we dedicate a complete analysis over its principal features to gain a better understanding of the algorithms and combinatorial problems.

In the first experiment we realized, we observed that the behavior of our framework is similar to the current reference in the state-of-the-art. The outcome of this first experiment was relatively clear: this confirmed that our framework was capable of generating high quality solutions. Therefore, it was evident that the following level was to create generic solution for different combinatorial problems. We have performed a complete redesign inspired by the main characteristics of ant colony optimization algorithms, building a new algorithm that also provides alternative methods proposed in the literature, and eventually archiving a generic configuration that can deal with whatever combinatorial problem with base on a permutation.

We did not have the opportunity of improving the result of the different proposed problems, but this thesis gives the chance to study how the ant colony optimization works over difference permutation problems from another point of view.

Appendix A

Representation of bounds

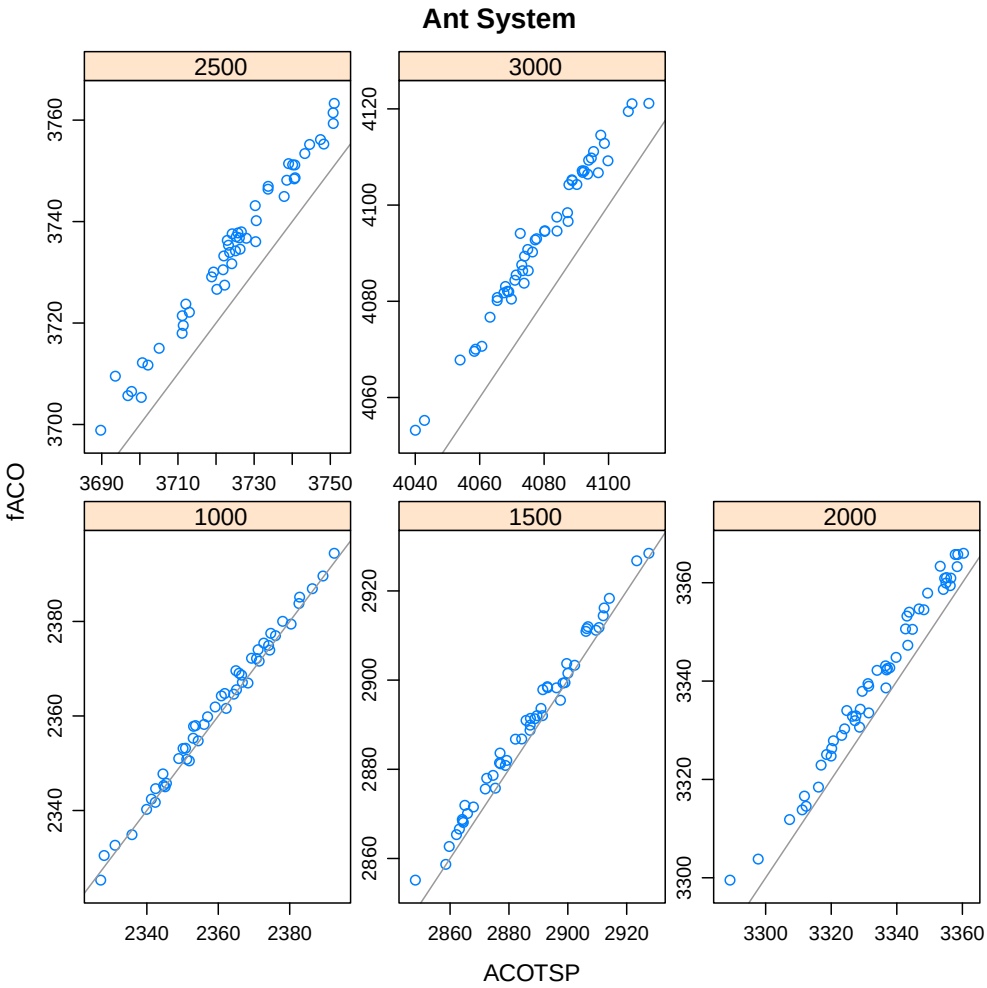
The following equations are proposed in [4, 14].

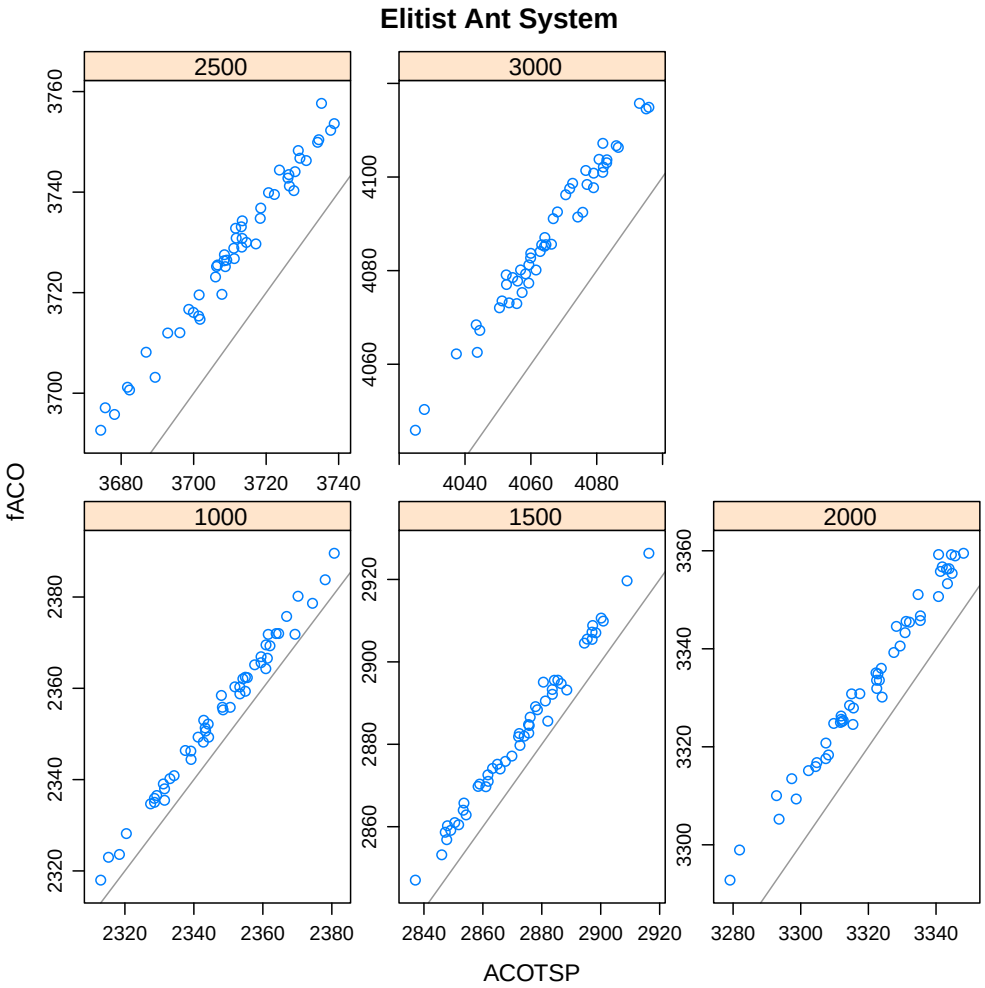
	A	B
bound lower		
A	0.0	
B	$\frac{\tau_{max}}{elected\ ants \cdot dimension}$	
C	$\frac{\tau_{max}(1 - \sqrt[permutation\ size]{0.05})}{(avg-1)\sqrt[permutation\ size]{0.05}}$	
bound upper		
A	$\frac{population\ size}{C^0}$	$population\ size \cdot C^0$
B	$\frac{population\ size + elitist\ ants}{\rho \cdot C^0}$	$population\ size + elitist\ ants \cdot \rho \cdot C^0$
C	$\frac{0.5 \cdot rank(rank-1)}{\rho \cdot C^0}$	$0.5 \cdot rank(rank - 1) \cdot \rho \cdot C^0$
D	$\frac{1}{\rho \cdot C^0}$	$\rho \cdot C^0$
E	$\frac{1}{permutation\ size \cdot C^0}$	$permutation\ size \cdot C^0$

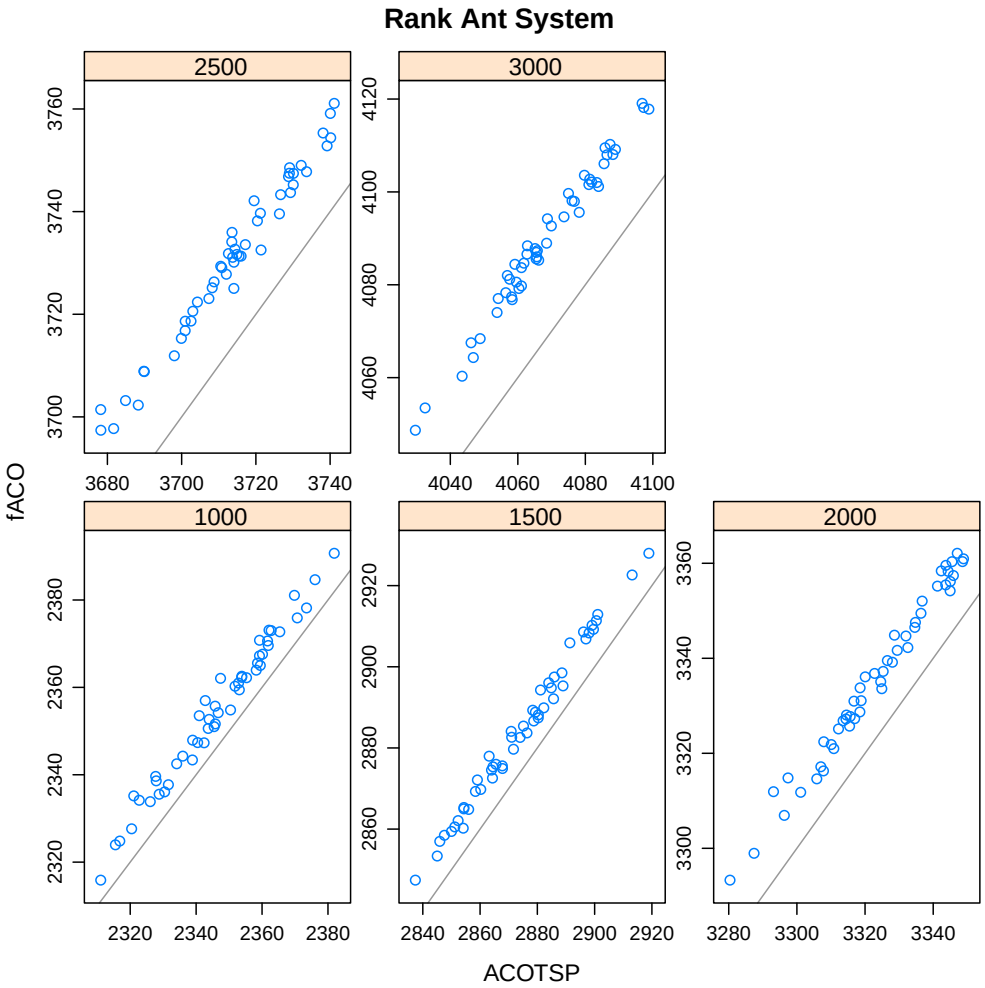
Where C^0 is the initial value of the objective function or 1.

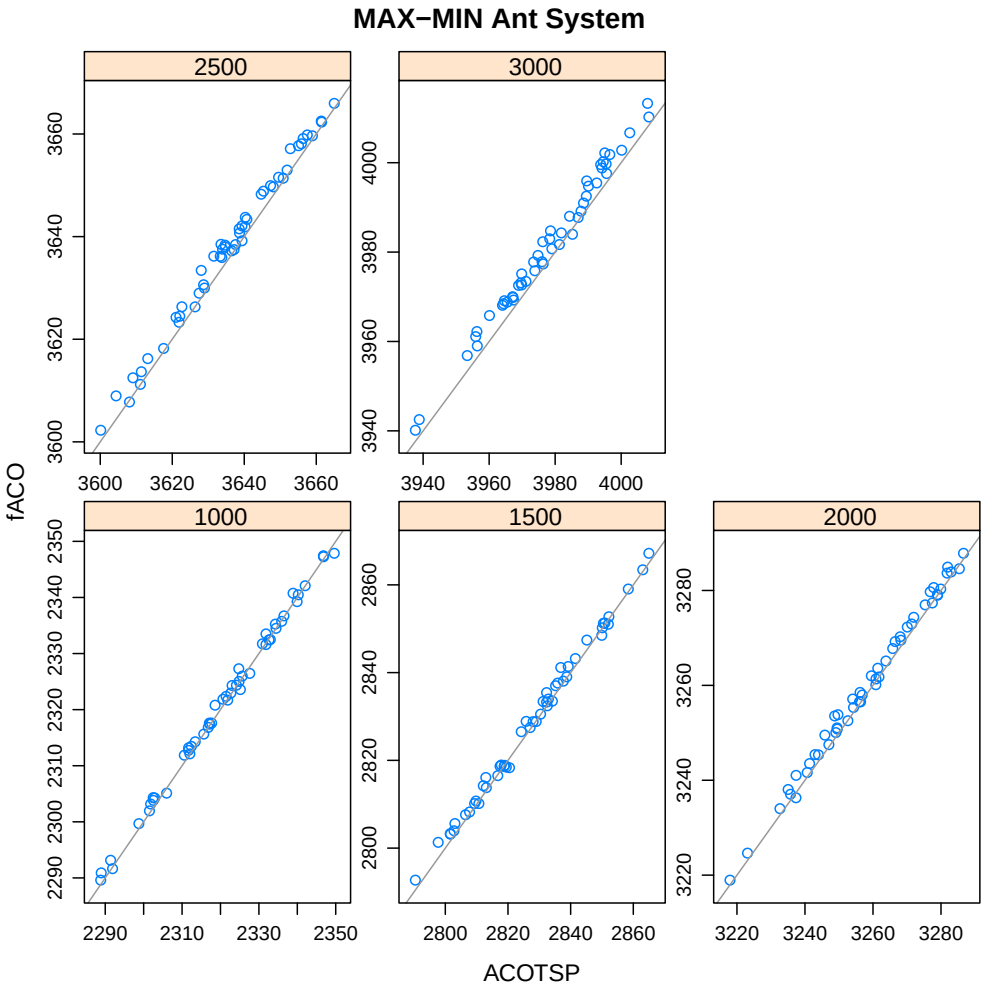
Appendix B

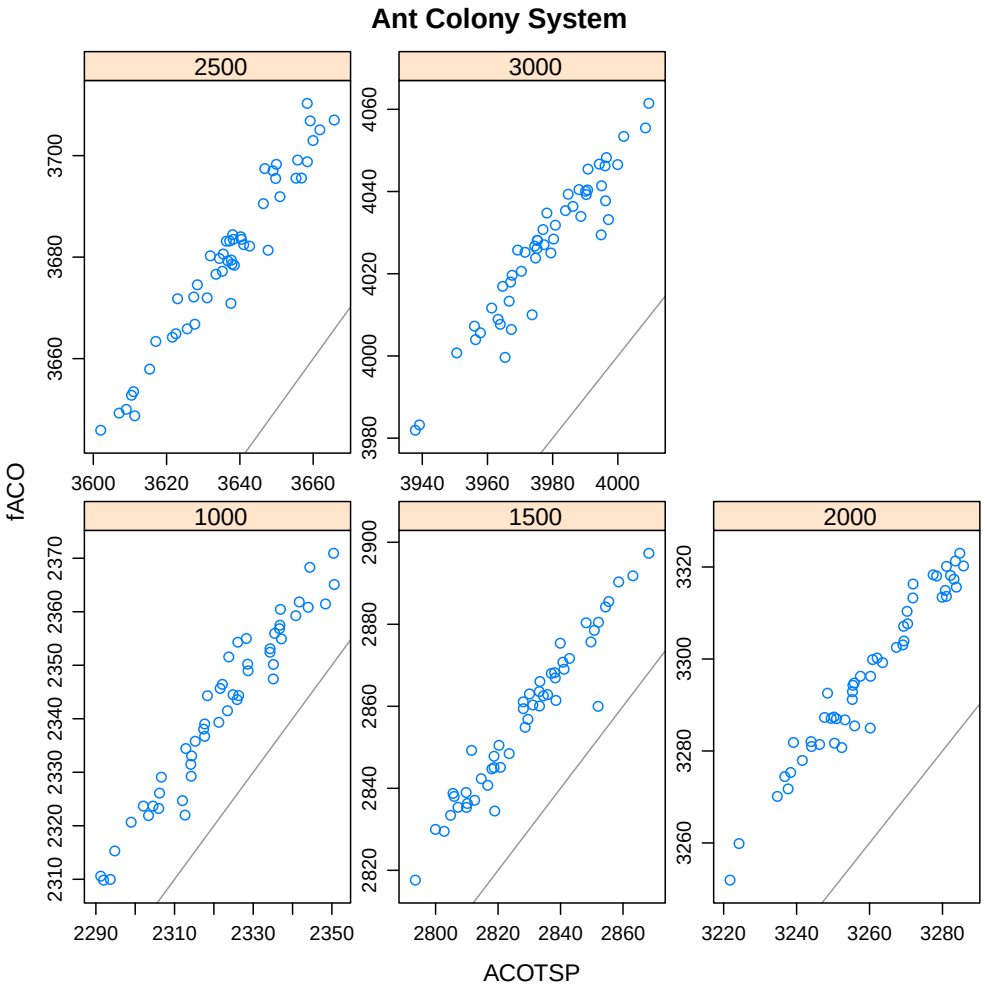
Solution quality compared to ACOTSP


Ant System
2500
3000
1000
1500
2000
fACO
ACOTSP
3700
3720
3740
3760
3690
3710
3730
3750
4060
4080
4100
4120
4040
4060
4080
4100
2340
2360
2380
2340
2360
2380
2860
2880
2900
2920
2860
2880
2900
2920
3300
3320
3340
3360
3300
3320
3340
3360


Elitist Ant System
2500
3000
1000
1500
2000
fACO
ACOTSP
3700
3720
3740
3760
4060
4080
4100
4040
2320
2340
2360
2380
2860
2880
2900
2920
2840
3300
3320
3340
3360
3280


Rank Ant System
2500
3000
1000
1500
2000
fACO
ACOTSP
3680 3700 3720 3740
3700 3720 3740 3760
4040 4060 4080 4100
4060 4080 4100 4120
2320 2340 2360 2380
2320 2340 2360 2380
2840 2860 2880 2900 2920
2860 2880 2900 2920
3280 3300 3320 3340
3300 3320 3340 3360

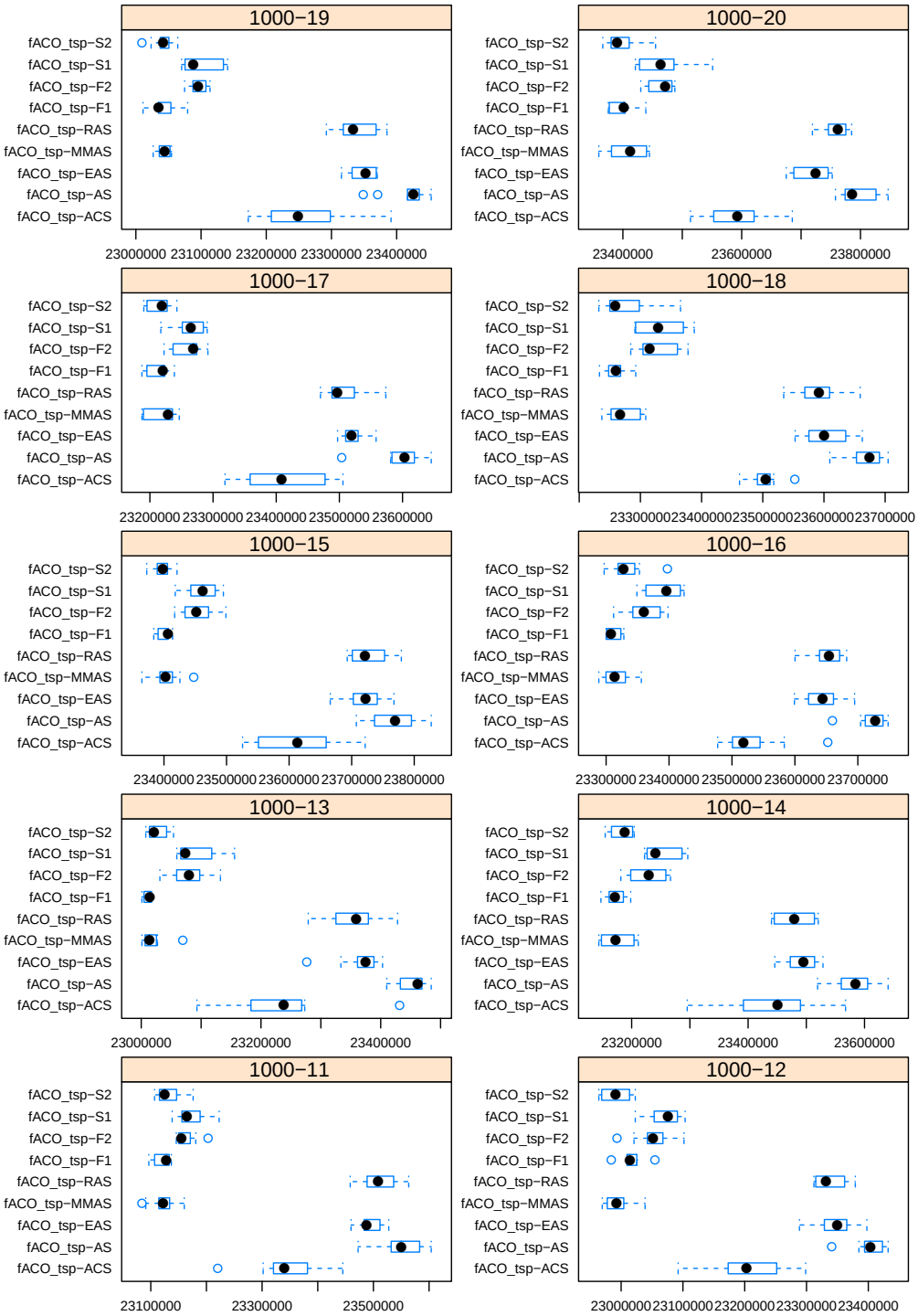

MAX−MIN Ant System
2500
3000
1000
1500
2000
fACO
ACOTSP
3600
3620
3640
3660
3940
3960
3980
4000
2290
2300
2310
2320
2330
2340
2350
2800
2820
2840
2860
3220
3240
3260
3280

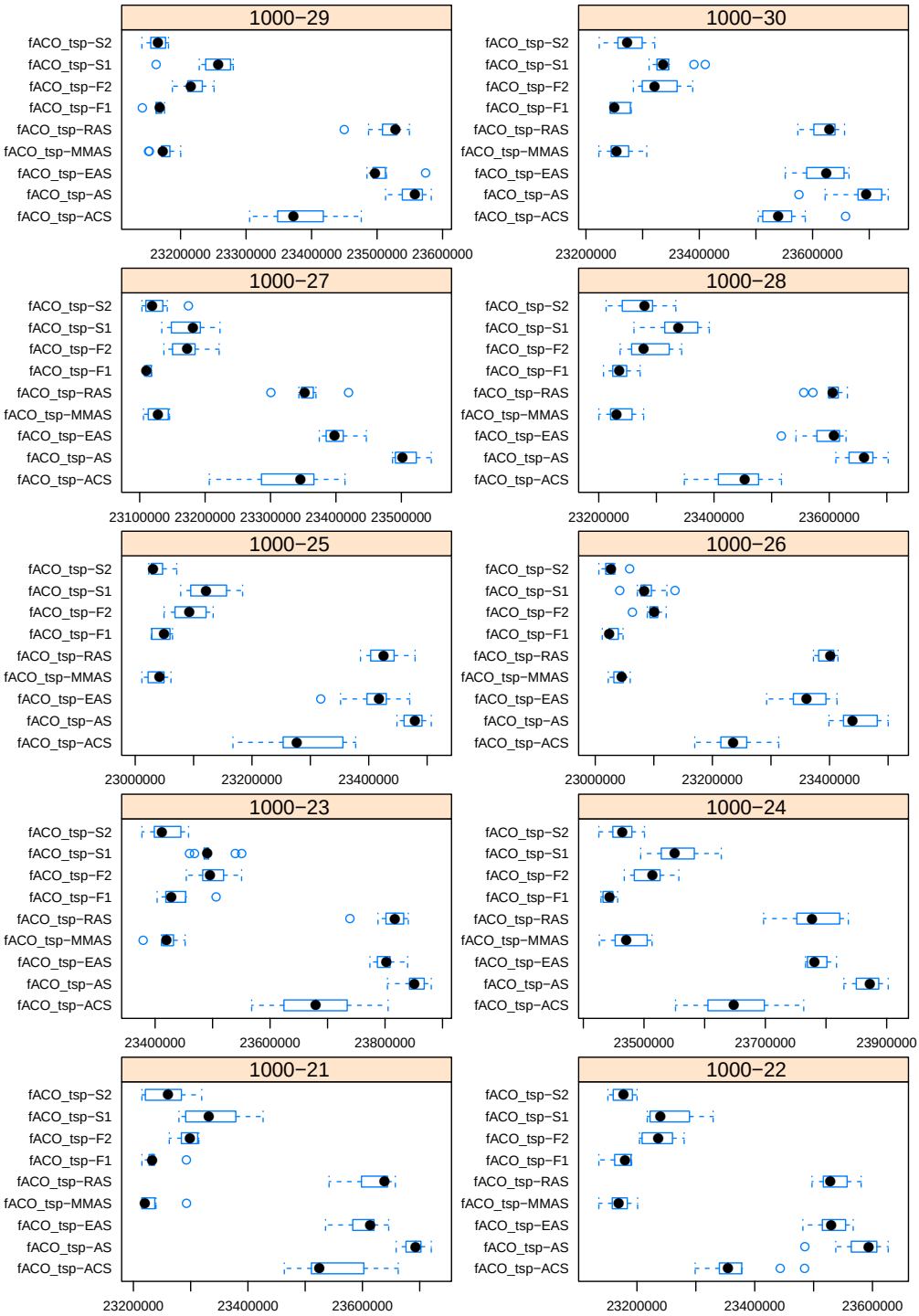

Ant Colony System
2500
3000
1000
1500
2000
fACO
ACOTSP
3600 3620 3640 3660
3660 3680 3700
3940 3960 3980 4000
3980 4000 4020 4040 4060
2290 2310 2330 2350
2310 2320 2330 2340 2350 2360 2370
2800 2820 2840 2860
2820 2840 2860 2880 2900
3220 3240 3260 3280
3260 3280 3300 3320

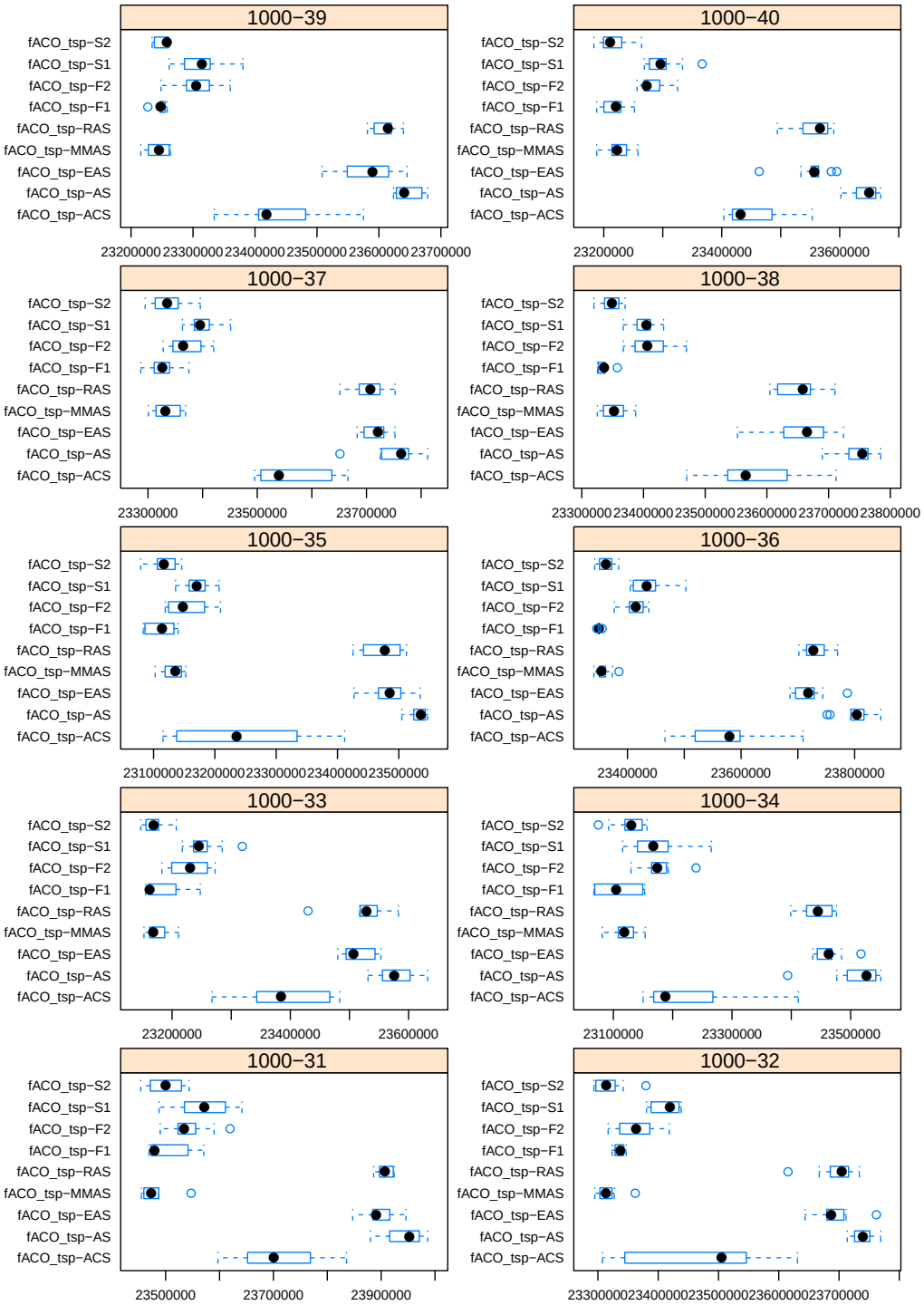
Appendix C

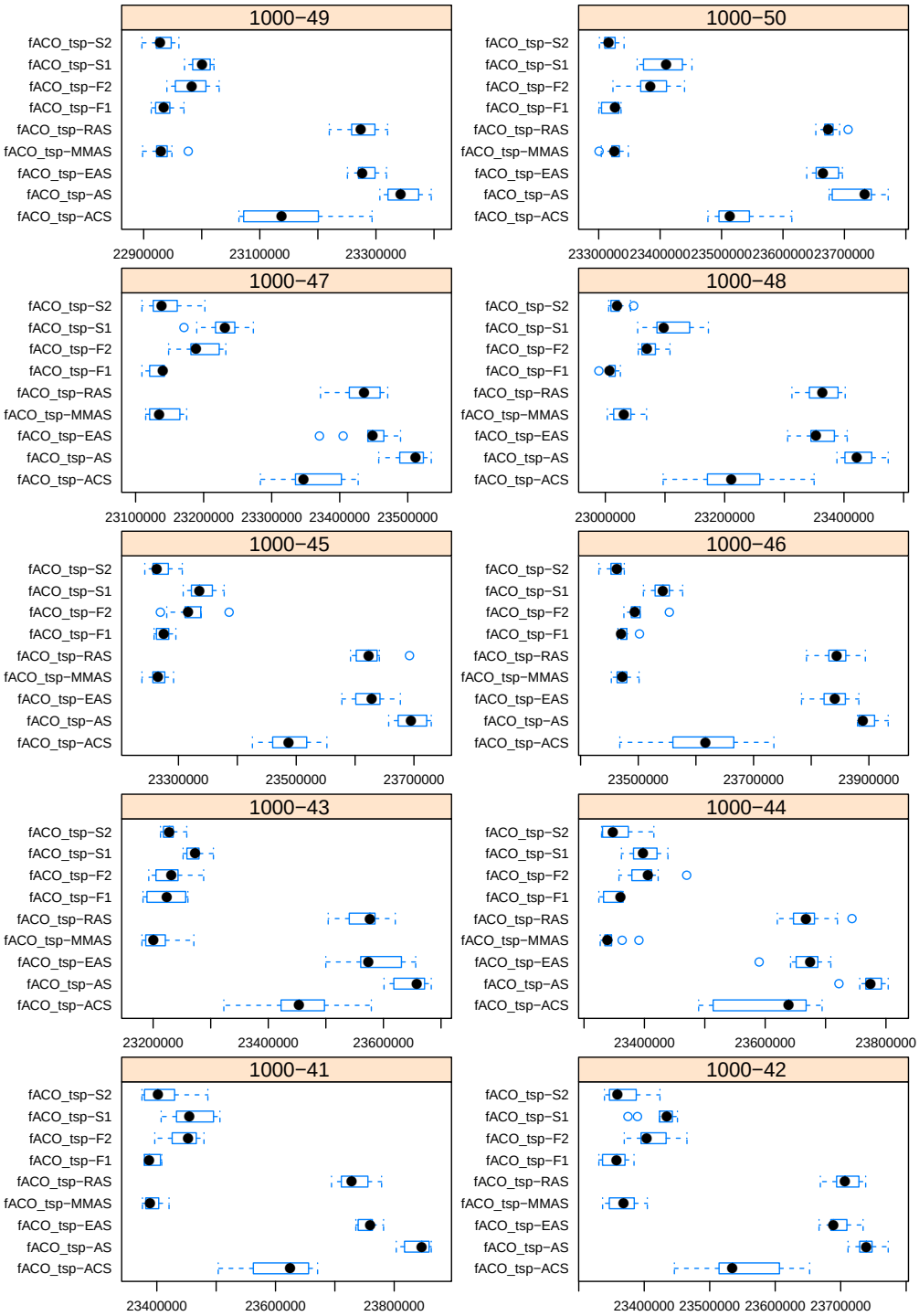
Results in detail

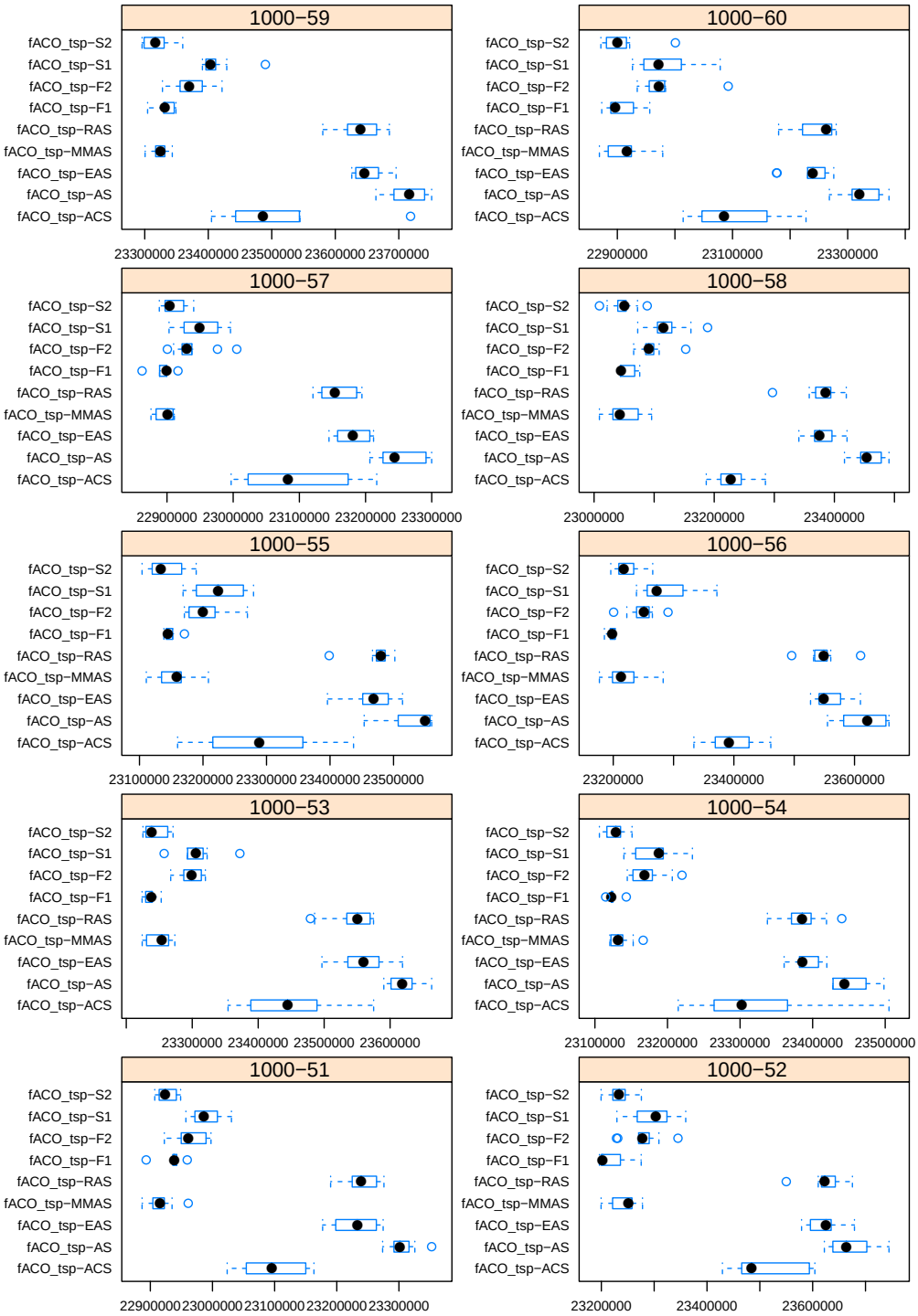
Traveling Salesman Problem

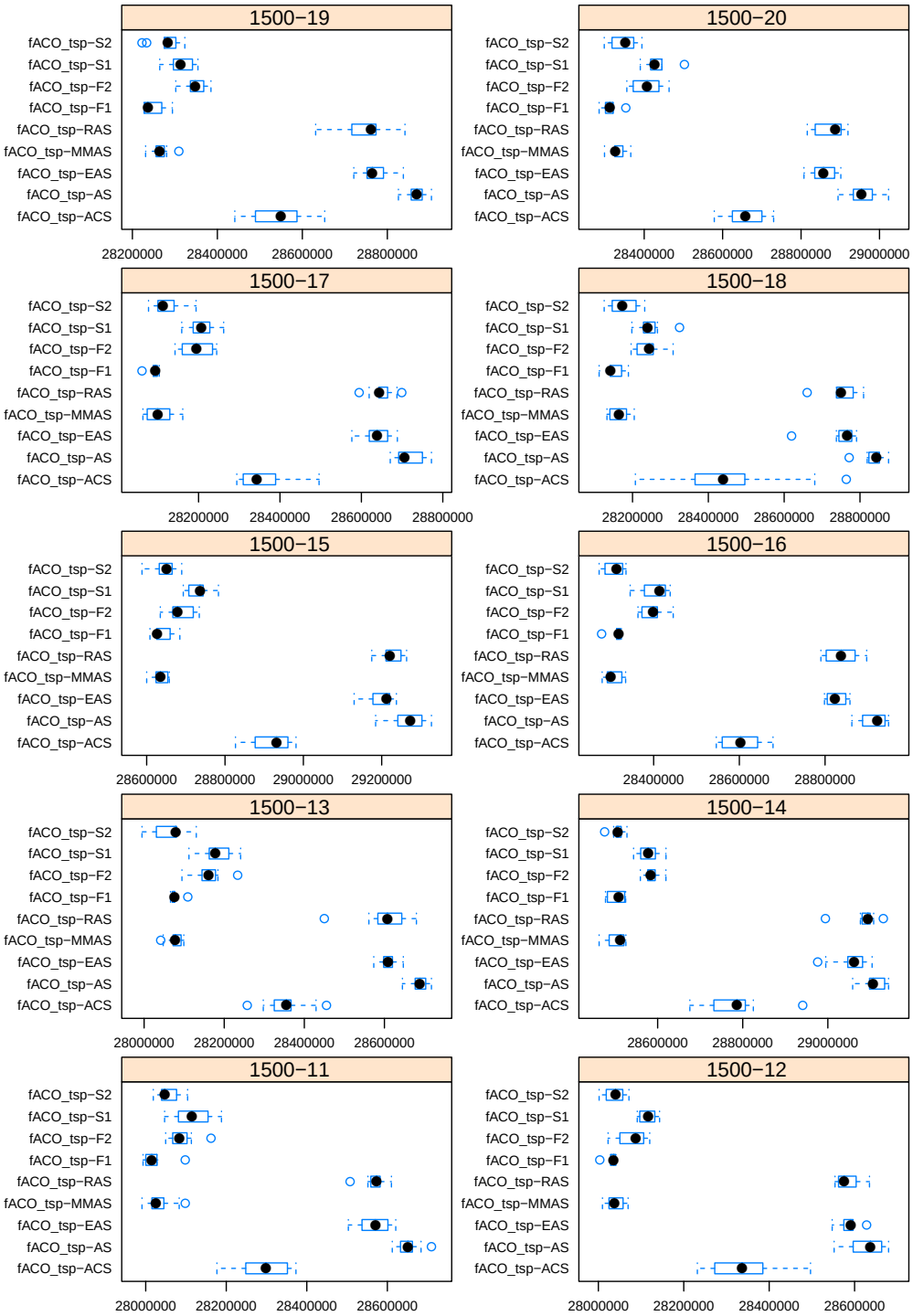

1000−19
fACO_tsp−S2
fACO_tsp−S1
fACO_tsp−F2
fACO_tsp−F1
fACO_tsp−RAS
fACO_tsp−MMAS
fACO_tsp−EAS
fACO_tsp−AS
fACO_tsp−ACS
23000000 23100000 23200000 23300000 23400000
1000−20
23400000 23600000 23800000
1000−17
23200000 23300000 23400000 23500000 23600000
1000−18
23300000 23400000 23500000 23600000 23700000
1000−15
23400000 23500000 23600000 23700000 23800000
1000−16
23300000 23400000 23500000 23600000 23700000
1000−13
23000000 23200000 23400000
1000−14
23200000 23400000 23600000
1000−11
23100000 23300000 23500000
1000−12
23000000 23100000 23200000 23300000 23400000

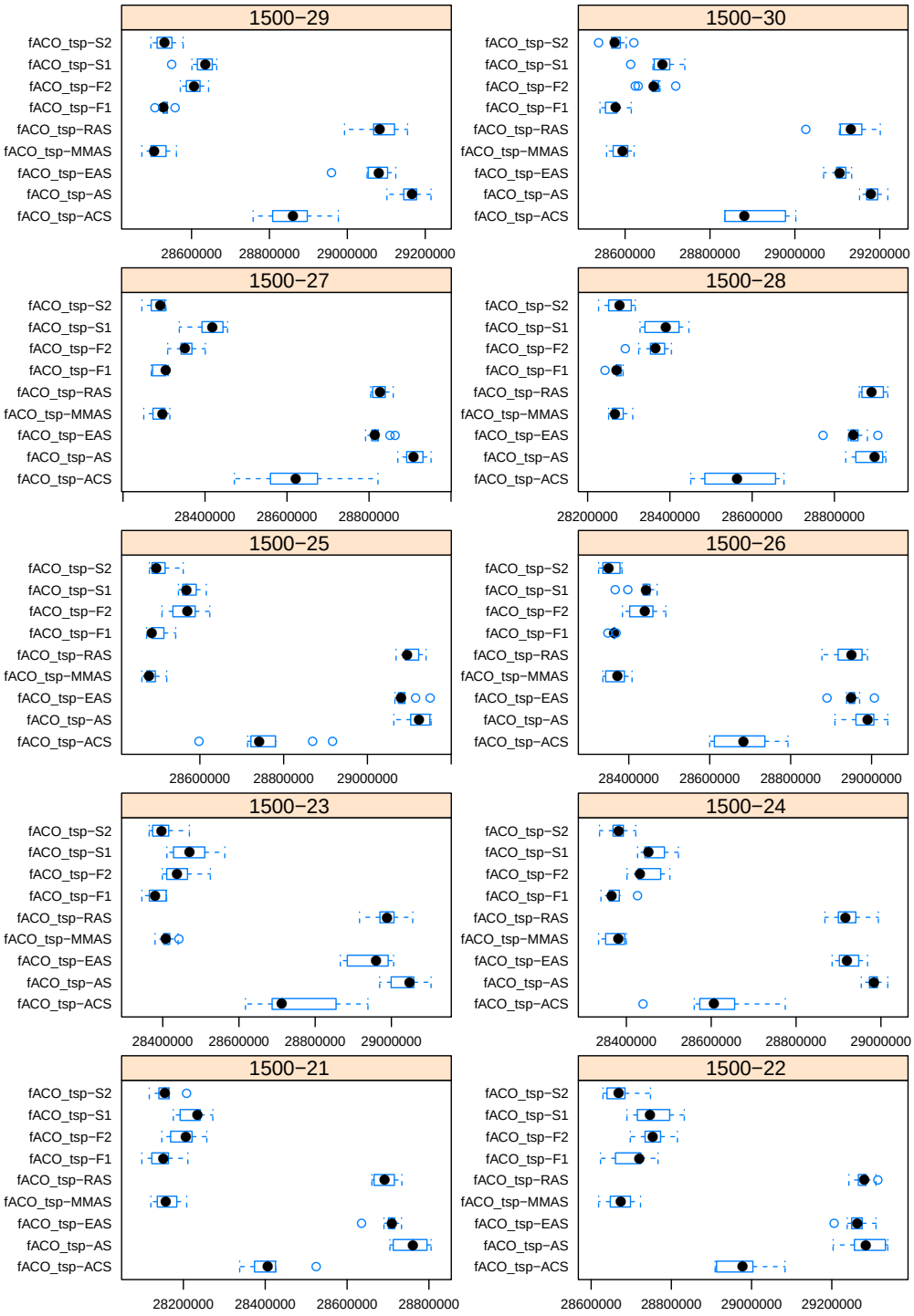

1000−29
1000−30
1000−27
1000−28
1000−25
1000−26
1000−23
1000−24
1000−21
1000−22
fACO_tsp−S2
fACO_tsp−S1
fACO_tsp−F2
fACO_tsp−F1
fACO_tsp−RAS
fACO_tsp−MMAS
fACO_tsp−EAS
fACO_tsp−AS
fACO_tsp−ACS
23200000 23300000 23400000 23500000 23600000
23200000 23400000 23600000
23100000 23200000 23300000 23400000 23500000
23200000 23400000 23600000
23000000 23200000 23400000
23000000 23200000 23400000
23400000 23600000 23800000
23500000 23700000 23900000
23200000 23400000 23600000
23200000 23400000 23600000

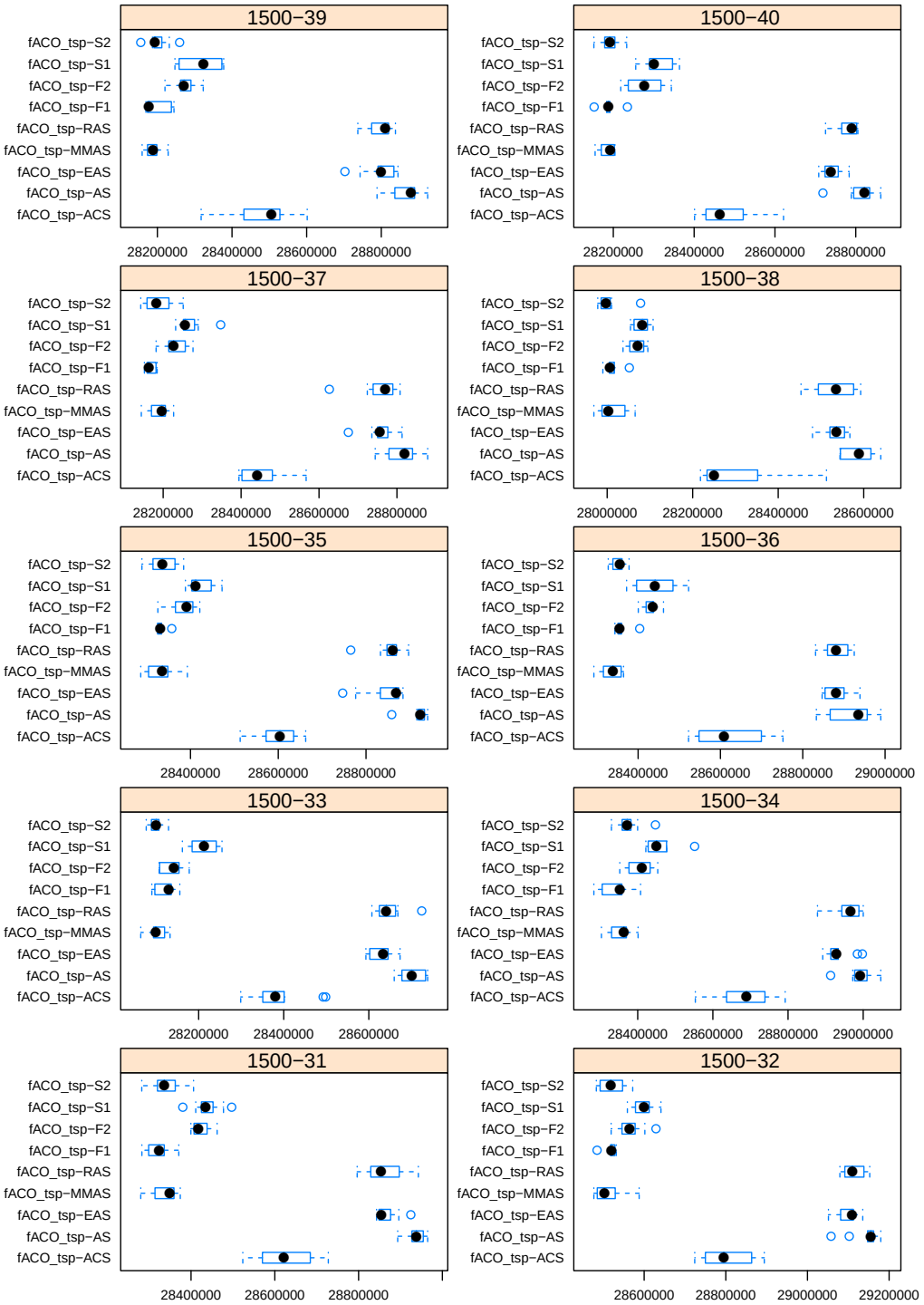

1000−39
fACO_tsp−S2
fACO_tsp−S1
fACO_tsp−F2
fACO_tsp−F1
fACO_tsp−RAS
fACO_tsp−MMAS
fACO_tsp−EAS
fACO_tsp−AS
fACO_tsp−ACS
23200000 23300000 23400000 23500000 23600000 23700000
1000−40
23200000 23400000 23600000
1000−37
23300000 23500000 23700000
1000−38
23300000 23400000 23500000 23600000 23700000 23800000
1000−35
23100000 23200000 23300000 23400000 23500000
1000−36
23400000 23600000 23800000
1000−33
23200000 23400000 23600000
1000−34
23100000 23300000 23500000
1000−31
23500000 23700000 23900000
1000−32
23300000 23400000 23500000 23600000 23700000

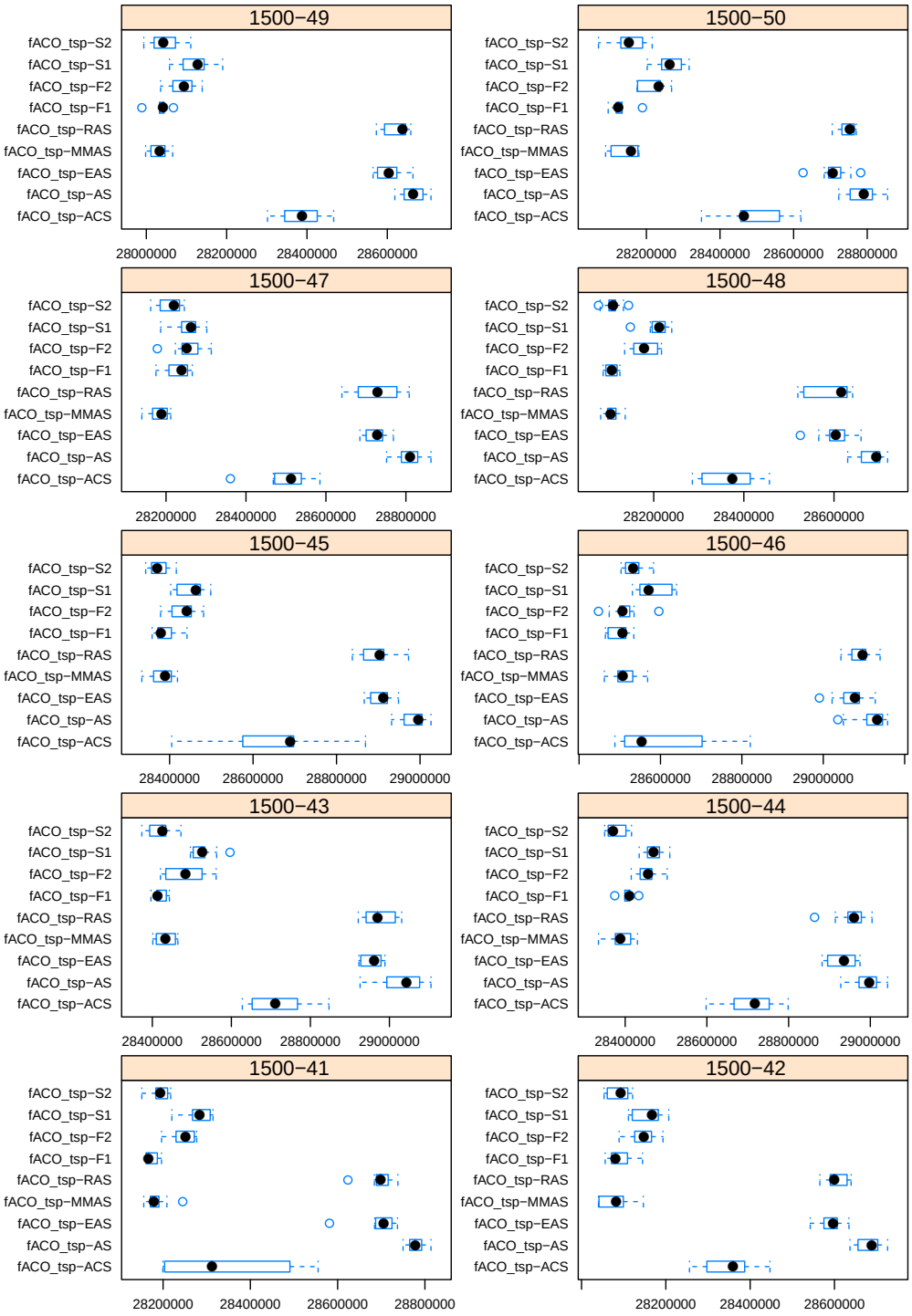

1000−49
1000−50
1000−47
1000−48
1000−45
1000−46
1000−43
1000−44
1000−41
1000−42
fACO_tsp−S2
fACO_tsp−S1
fACO_tsp−F2
fACO_tsp−F1
fACO_tsp−RAS
fACO_tsp−MMAS
fACO_tsp−EAS
fACO_tsp−AS
fACO_tsp−ACS
22900000 23100000 23300000
23300000 23400000 23500000 23600000 23700000
23100000 23200000 23300000 23400000 23500000
23000000 23200000 23400000
23300000 23500000 23700000
23500000 23700000 23900000
23200000 23400000 23600000
23400000 23600000 23800000
23400000 23600000 23800000
23400000 23500000 23600000 23700000

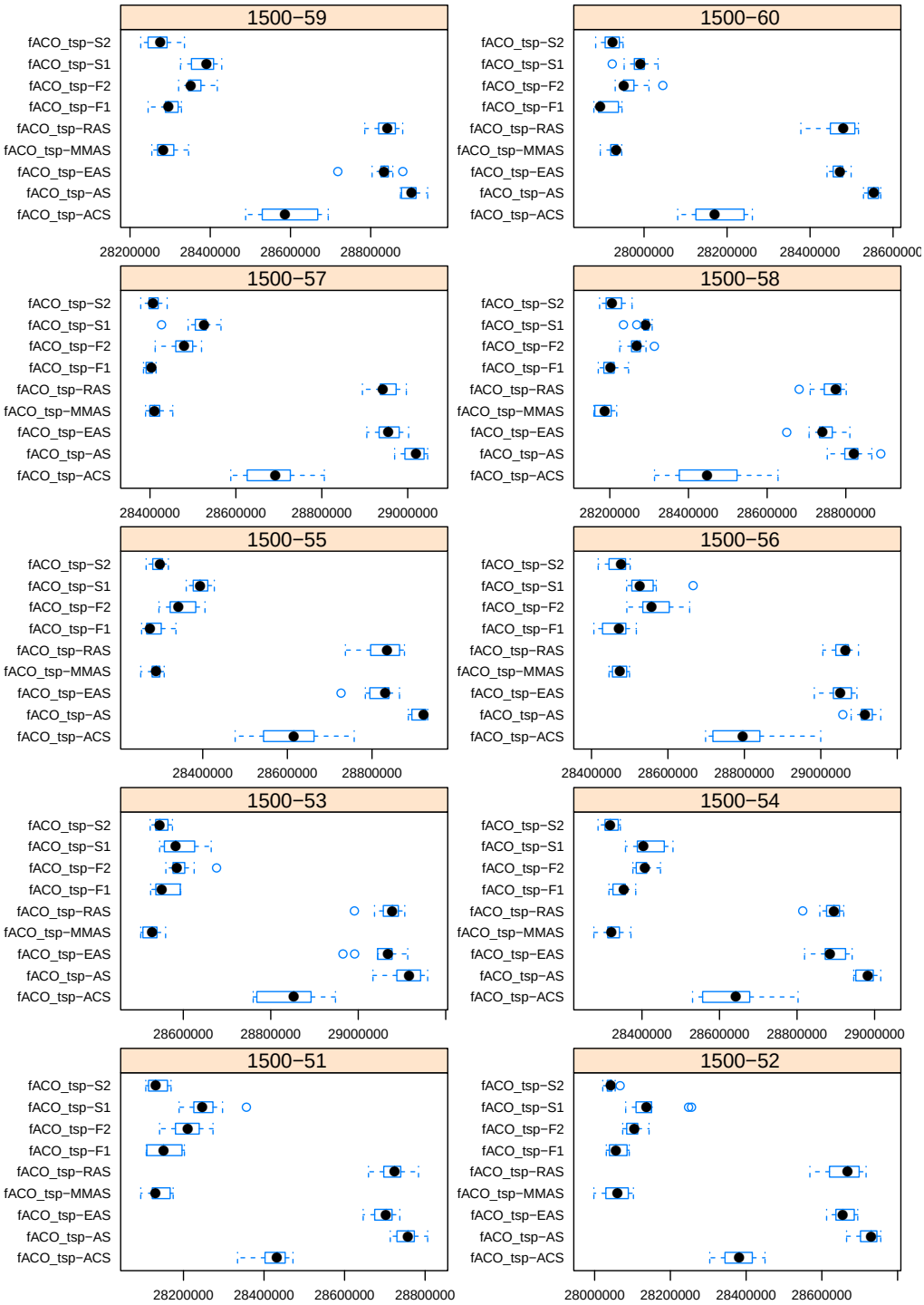

1000−59
1000−60
1000−57
1000−58
1000−55
1000−56
1000−53
1000−54
1000−51
1000−52
fACO_tsp−S2
fACO_tsp−S1
fACO_tsp−F2
fACO_tsp−F1
fACO_tsp−RAS
fACO_tsp−MMAS
fACO_tsp−EAS
fACO_tsp−AS
fACO_tsp−ACS
23300000 23400000 23500000 23600000 23700000
22900000 23100000 23300000
22900000 23000000 23100000 23200000 23300000
23000000 23200000 23400000
23100000 23200000 23300000 23400000 23500000
23200000 23400000 23600000
23300000 23400000 23500000 23600000
23100000 23200000 23300000 23400000 23500000
22900000 23000000 23100000 23200000 23300000
23200000 23400000 23600000

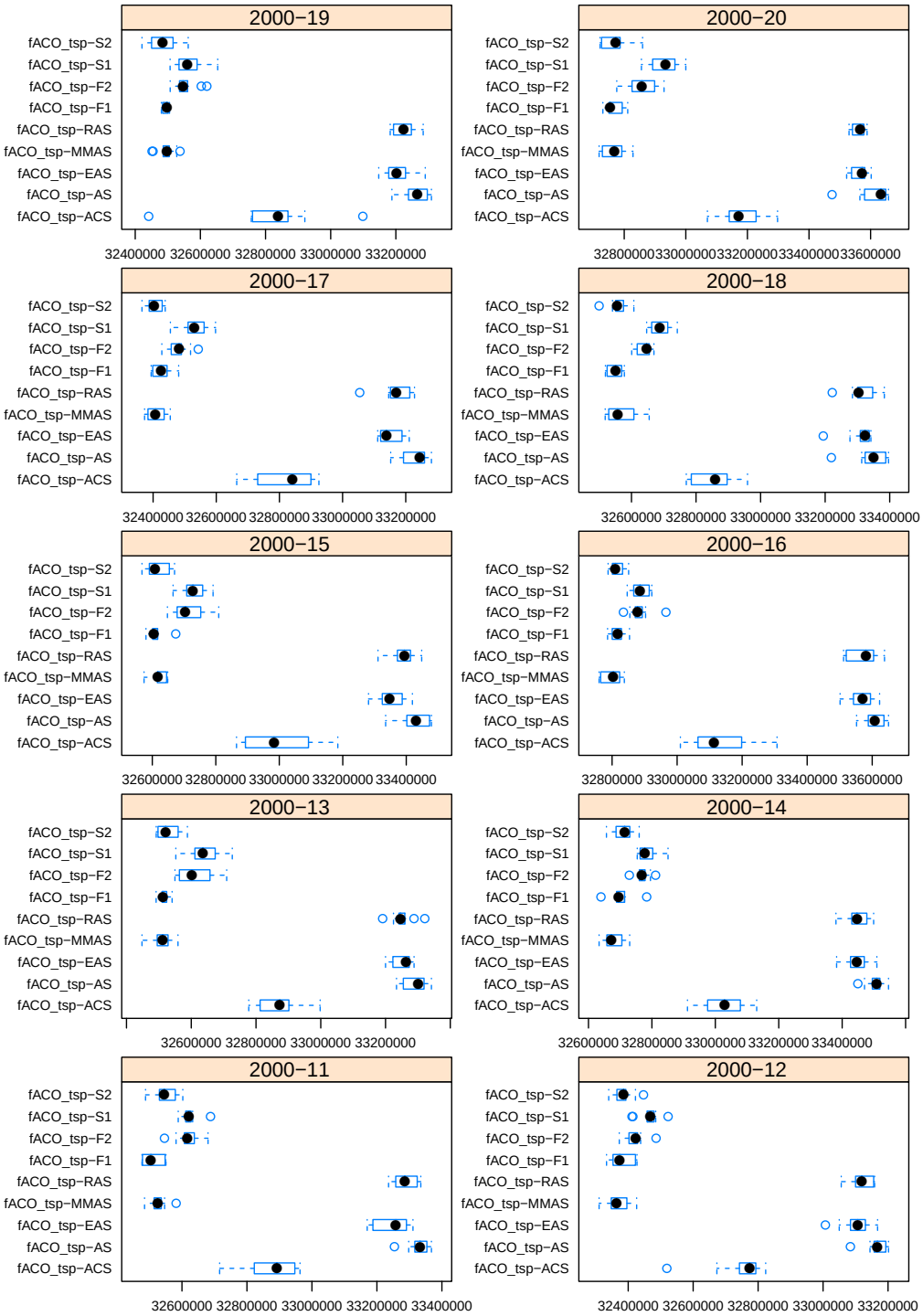

1500−19
1500−20
1500−17
1500−18
1500−15
1500−16
1500−13
1500−14
1500−11
1500−12
fACO_tsp−S2
fACO_tsp−S1
fACO_tsp−F2
fACO_tsp−F1
fACO_tsp−RAS
fACO_tsp−MMAS
fACO_tsp−EAS
fACO_tsp−AS
fACO_tsp−ACS
28200000
28400000
28600000
28800000
29000000
29200000

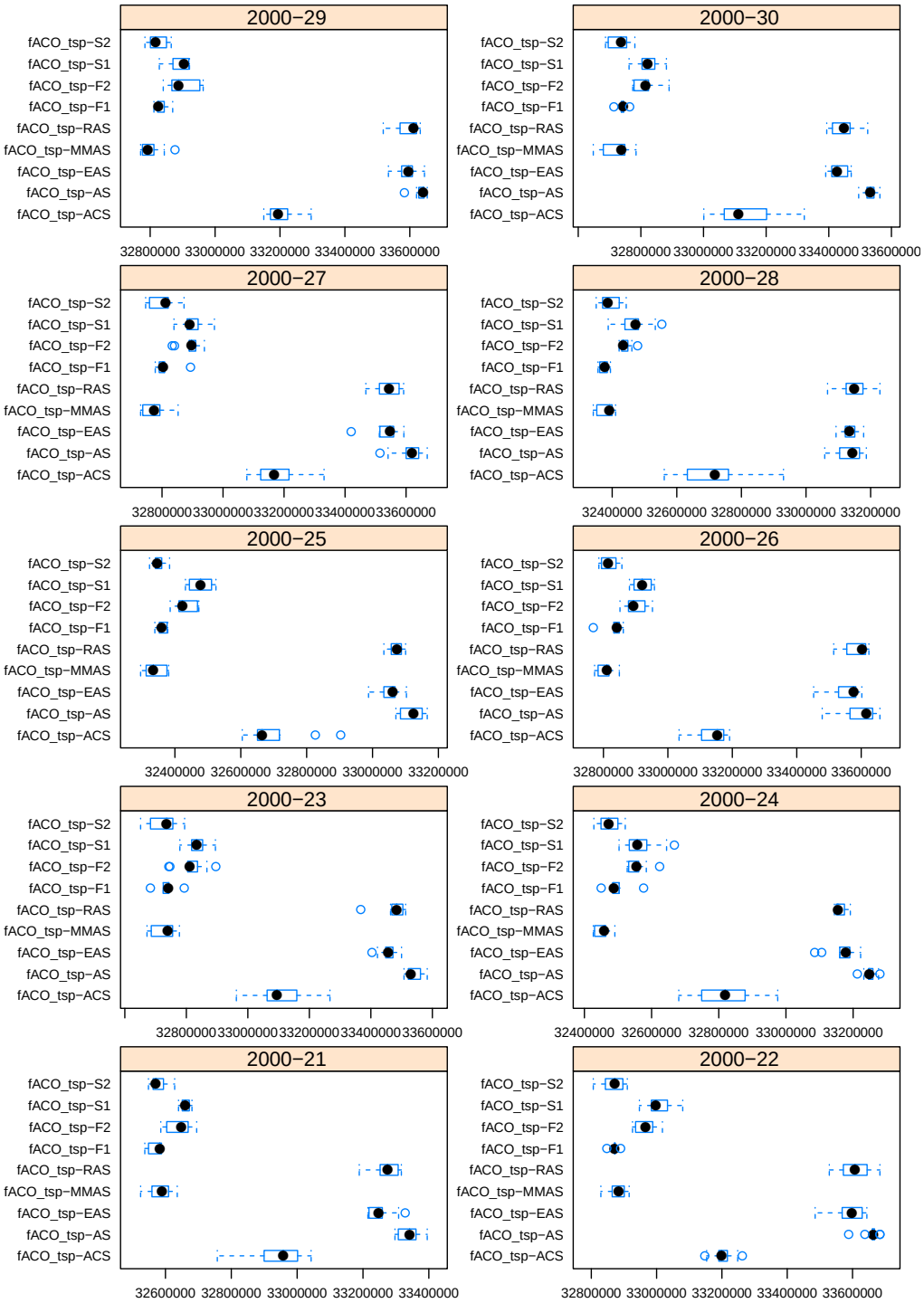

1500−29
1500−30
1500−27
1500−28
1500−25
1500−26
1500−23
1500−24
1500−21
1500−22
fACO_tsp−S2
fACO_tsp−S1
fACO_tsp−F2
fACO_tsp−F1
fACO_tsp−RAS
fACO_tsp−MMAS
fACO_tsp−EAS
fACO_tsp−AS
fACO_tsp−ACS
28600000
28800000
29000000
29200000
28400000
28200000

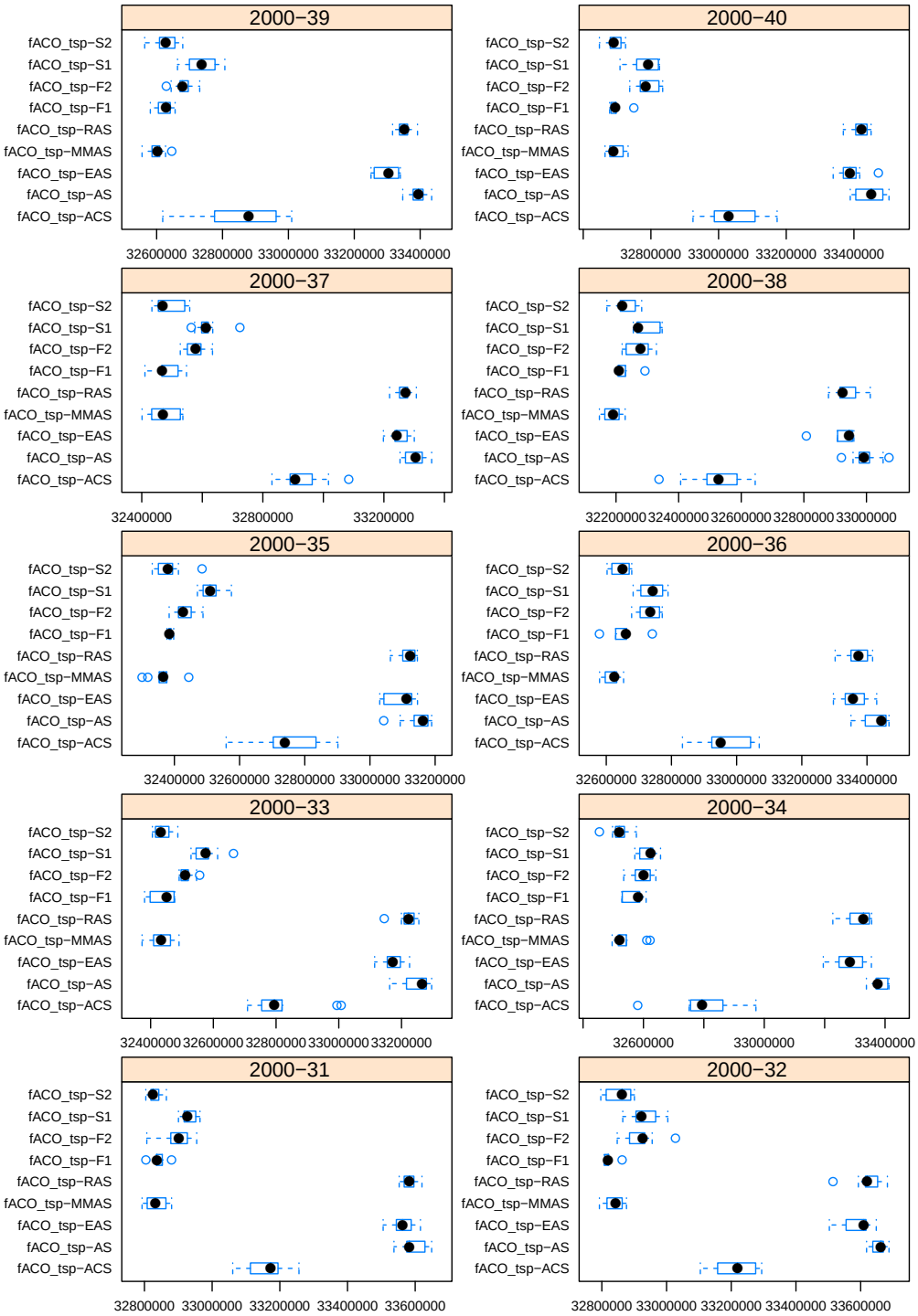

1500−39
1500−40
1500−37
1500−38
1500−35
1500−36
1500−33
1500−34
1500−31
1500−32
fACO_tsp−S2
fACO_tsp−S1
fACO_tsp−F2
fACO_tsp−F1
fACO_tsp−RAS
fACO_tsp−MMAS
fACO_tsp−EAS
fACO_tsp−AS
fACO_tsp−ACS
28200000
28400000
28600000
28800000
29000000
29200000

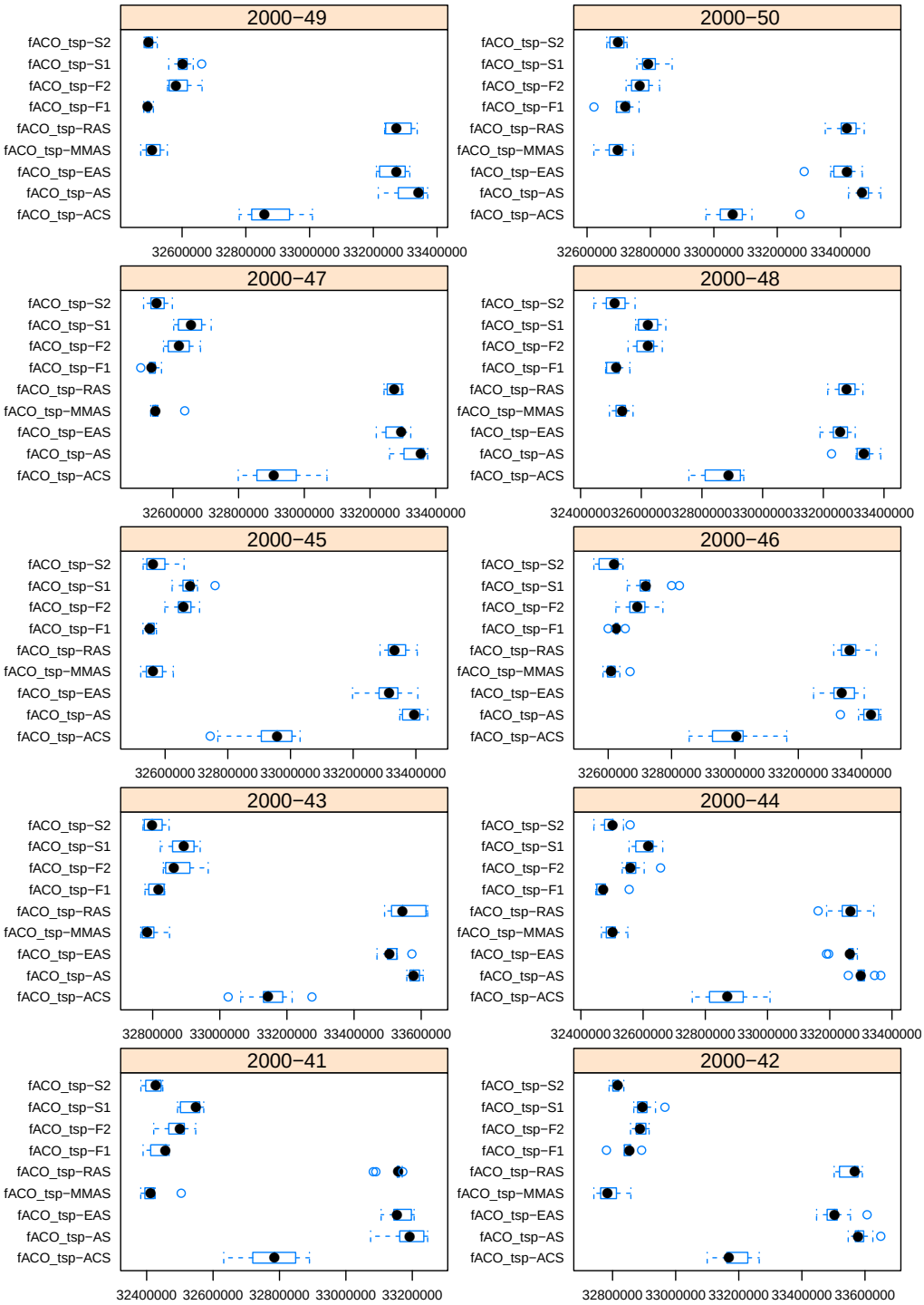

1500−49
1500−50
1500−47
1500−48
1500−45
1500−46
1500−43
1500−44
1500−41
1500−42
fACO_tsp−S2
fACO_tsp−S1
fACO_tsp−F2
fACO_tsp−F1
fACO_tsp−RAS
fACO_tsp−MMAS
fACO_tsp−EAS
fACO_tsp−AS
fACO_tsp−ACS
28000000
28200000
28400000
28600000
28800000
29000000

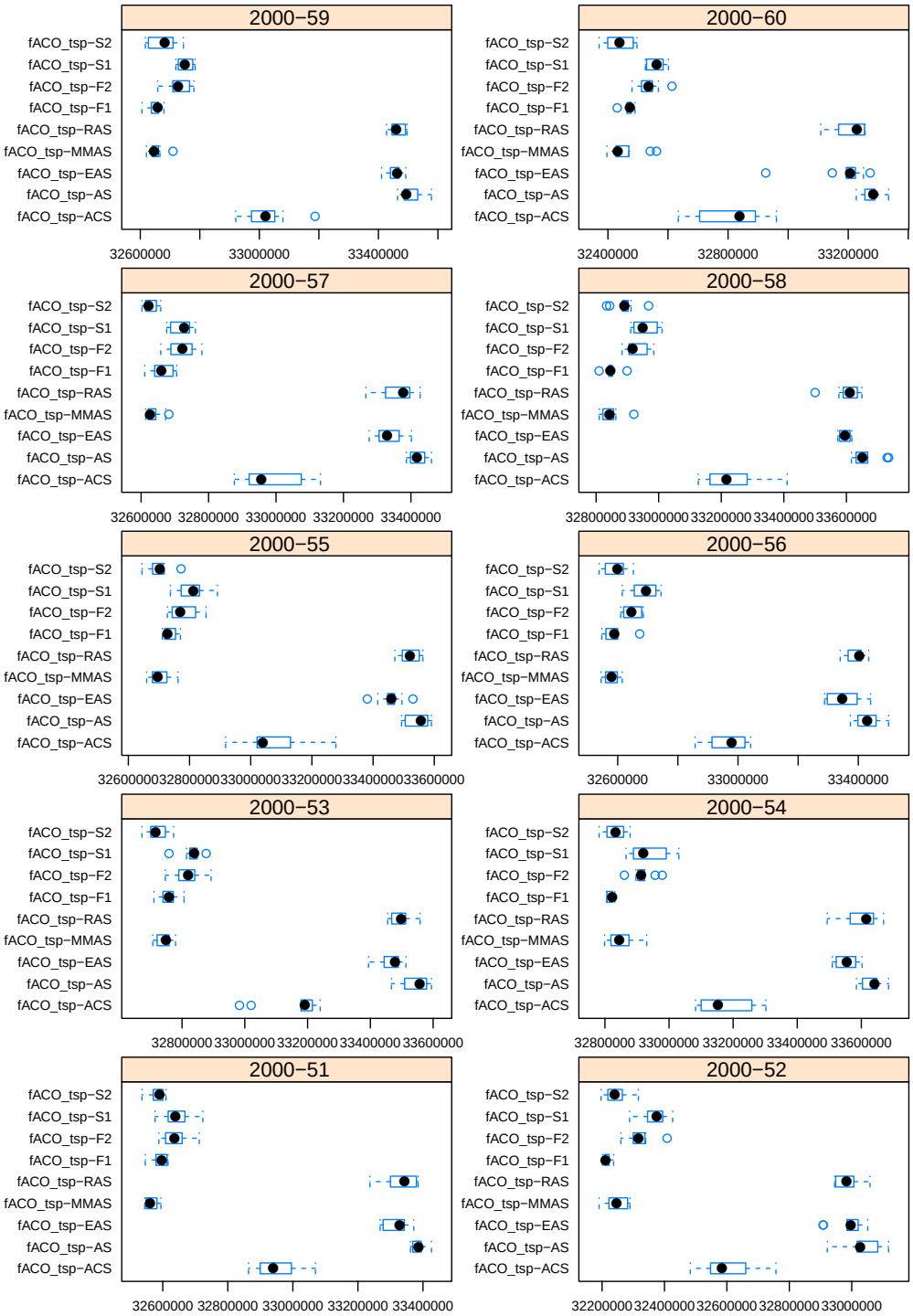

1500−59
1500−60
1500−57
1500−58
1500−55
1500−56
1500−53
1500−54
1500−51
1500−52
fACO_tsp−S2
fACO_tsp−S1
fACO_tsp−F2
fACO_tsp−F1
fACO_tsp−RAS
fACO_tsp−MMAS
fACO_tsp−EAS
fACO_tsp−AS
fACO_tsp−ACS
28200000
28400000
28600000
28800000
28000000
28200000
28400000
28600000
28400000
28600000
28800000
29000000
28200000
28400000
28600000
28800000
28400000
28600000
28800000
28400000
28600000
28800000
29000000
28600000
28800000
29000000
28400000
28600000
28800000
29000000
28200000
28400000
28600000
28800000
28000000
28200000
28400000
28600000

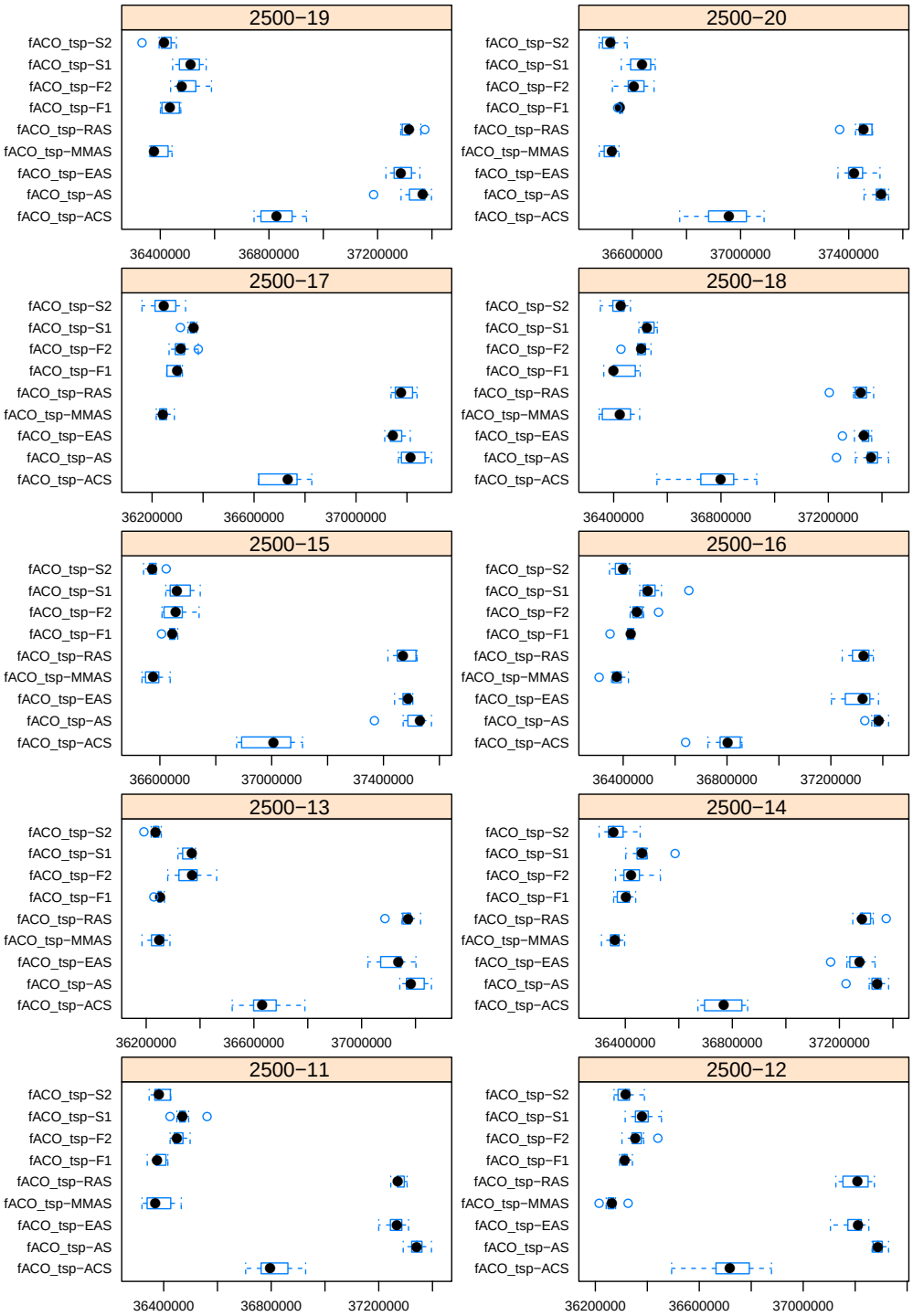

2000−19
2000−20
2000−17
2000−18
2000−15
2000−16
2000−13
2000−14
2000−11
2000−12
fACO_tsp−S2
fACO_tsp−S1
fACO_tsp−F2
fACO_tsp−F1
fACO_tsp−RAS
fACO_tsp−MMAS
fACO_tsp−EAS
fACO_tsp−AS
fACO_tsp−ACS
32400000 32600000 32800000 33000000 33200000
32800000 33000000 33200000 33400000 33600000
32400000 32600000 32800000 33000000 33200000
32600000 32800000 33000000 33200000 33400000
32600000 32800000 33000000 33200000 33400000
32800000 33000000 33200000 33400000 33600000
32600000 32800000 33000000 33200000
32600000 32800000 33000000 33200000 33400000
32600000 32800000 33000000 33200000 33400000
32400000 32600000 32800000 33000000 33200000

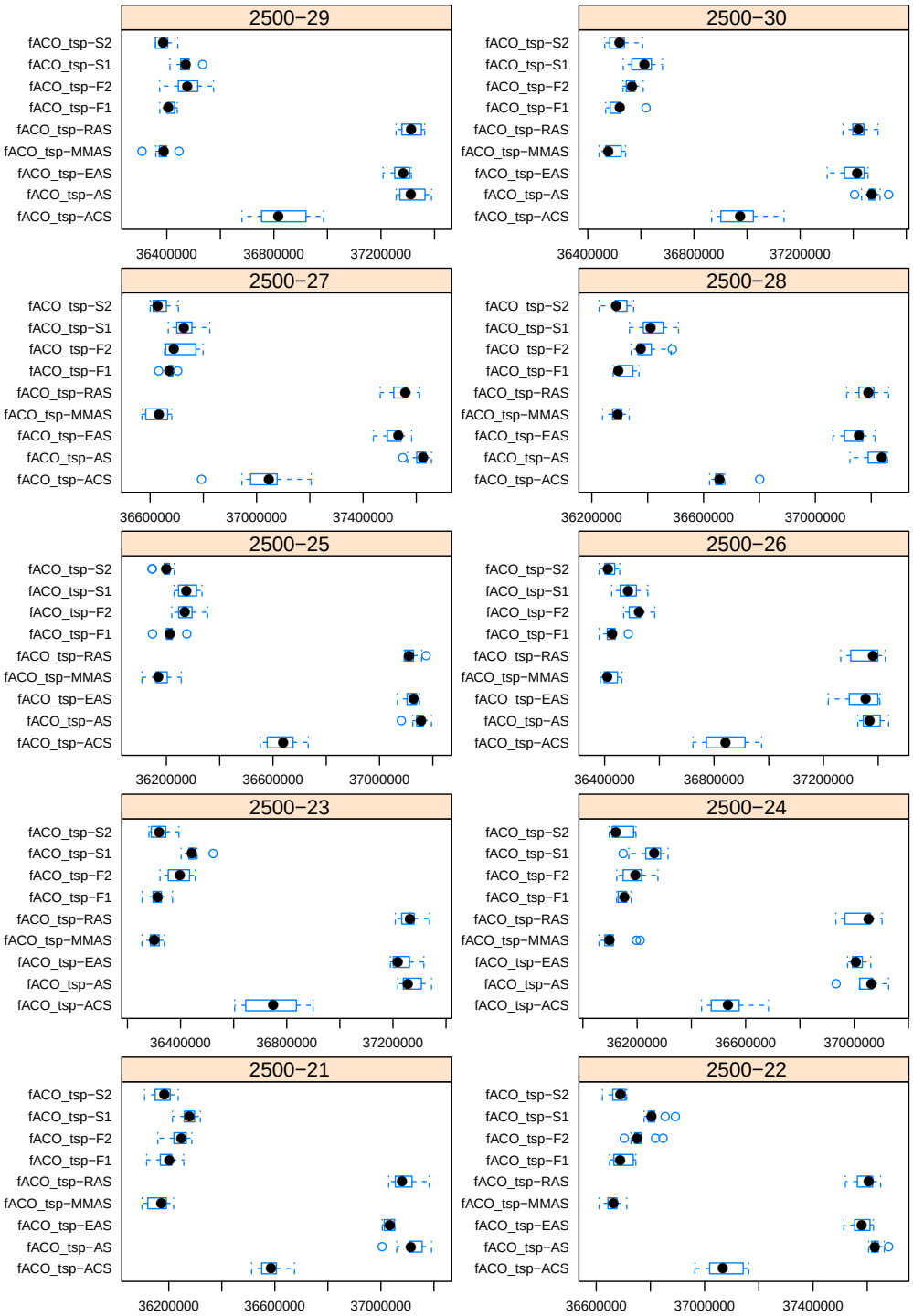

2000−29
2000−30
2000−27
2000−28
2000−25
2000−26
2000−23
2000−24
2000−21
2000−22
fACO_tsp−S2
fACO_tsp−S1
fACO_tsp−F2
fACO_tsp−F1
fACO_tsp−RAS
fACO_tsp−MMAS
fACO_tsp−EAS
fACO_tsp−AS
fACO_tsp−ACS
32800000 33000000 33200000 33400000 33600000
32400000 32600000 32800000 33000000 33200000

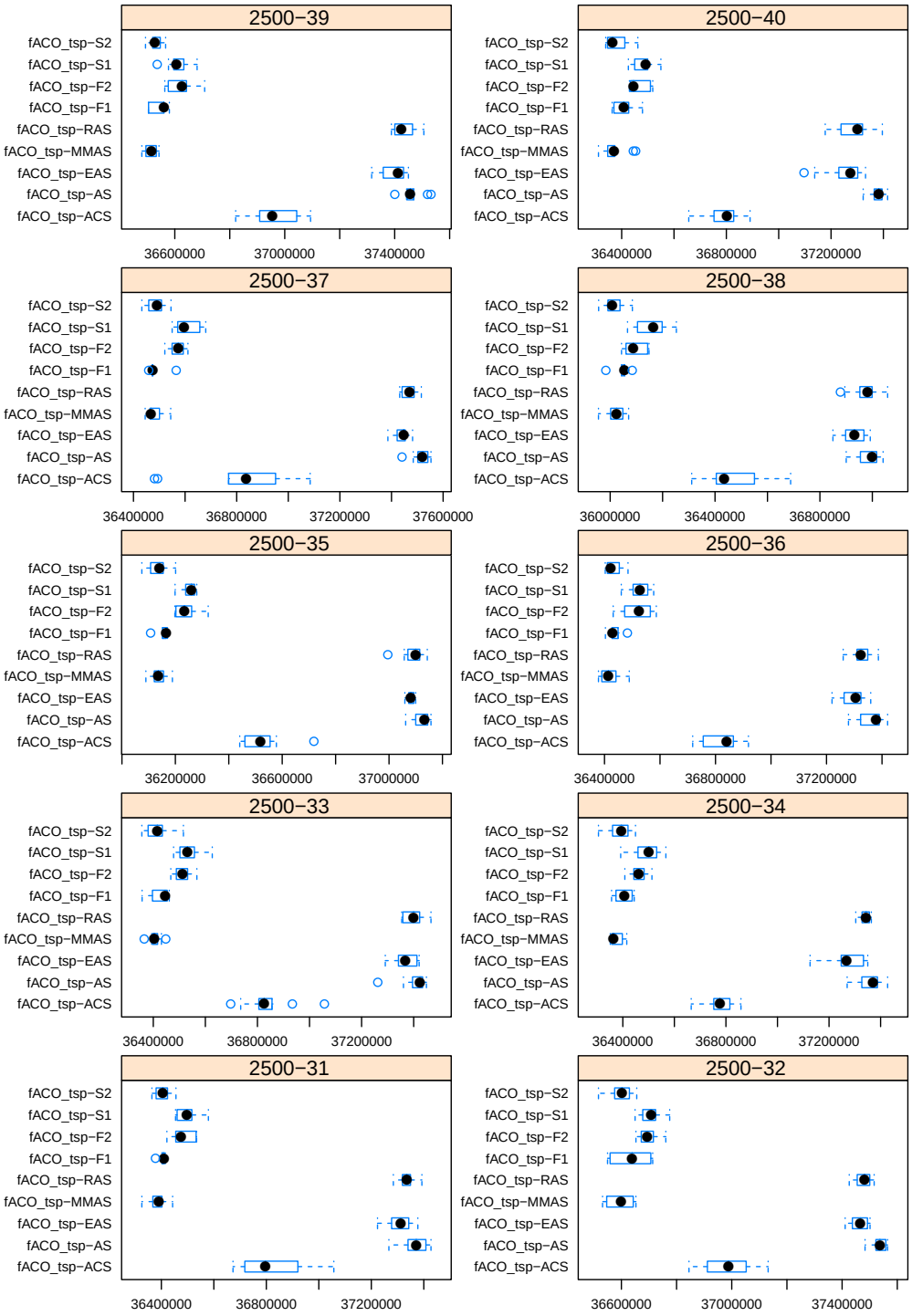

2000−39
2000−40
2000−37
2000−38
2000−35
2000−36
2000−33
2000−34
2000−31
2000−32
fACO_tsp−S2
fACO_tsp−S1
fACO_tsp−F2
fACO_tsp−F1
fACO_tsp−RAS
fACO_tsp−MMAS
fACO_tsp−EAS
fACO_tsp−AS
fACO_tsp−ACS
32600000 32800000 33000000 33200000 33400000
32800000 33000000 33200000 33400000
32400000 32800000 33200000
32200000 32400000 32600000 32800000 33000000
32400000 32600000 32800000 33000000 33200000
32600000 32800000 33000000 33200000 33400000
32400000 32600000 32800000 33000000 33200000
32600000 33000000 33400000
32800000 33000000 33200000 33400000 33600000
32800000 33000000 33200000 33400000 33600000

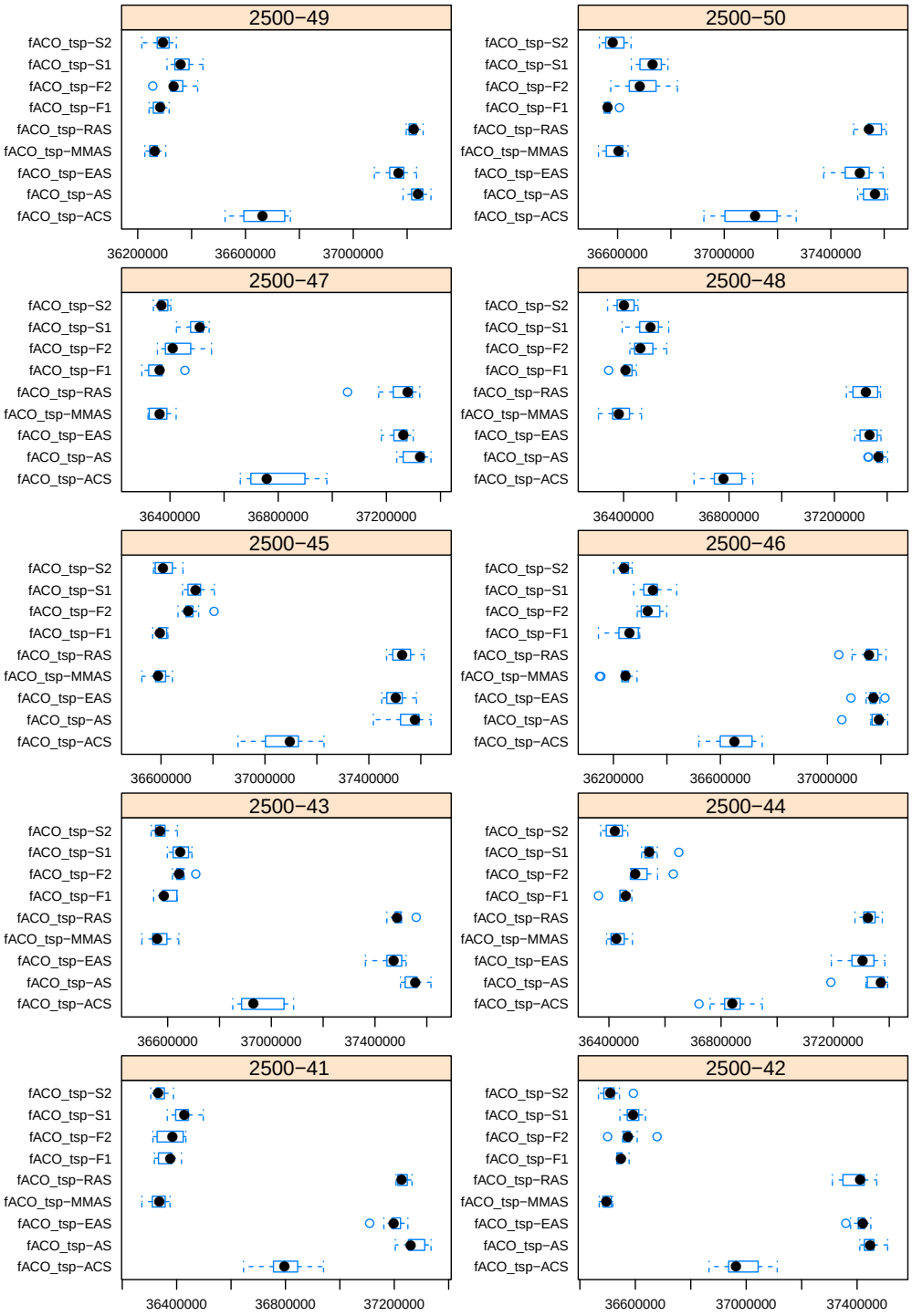

2000−49
2000−50
2000−47
2000−48
2000−45
2000−46
2000−43
2000−44
2000−41
2000−42
fACO_tsp−S2
fACO_tsp−S1
fACO_tsp−F2
fACO_tsp−F1
fACO_tsp−RAS
fACO_tsp−MMAS
fACO_tsp−EAS
fACO_tsp−AS
fACO_tsp−ACS
32600000 32800000 33000000 33200000 33400000
32400000 32600000 32800000 33000000 33200000 33400000
32800000 33000000 33200000 33400000 33600000

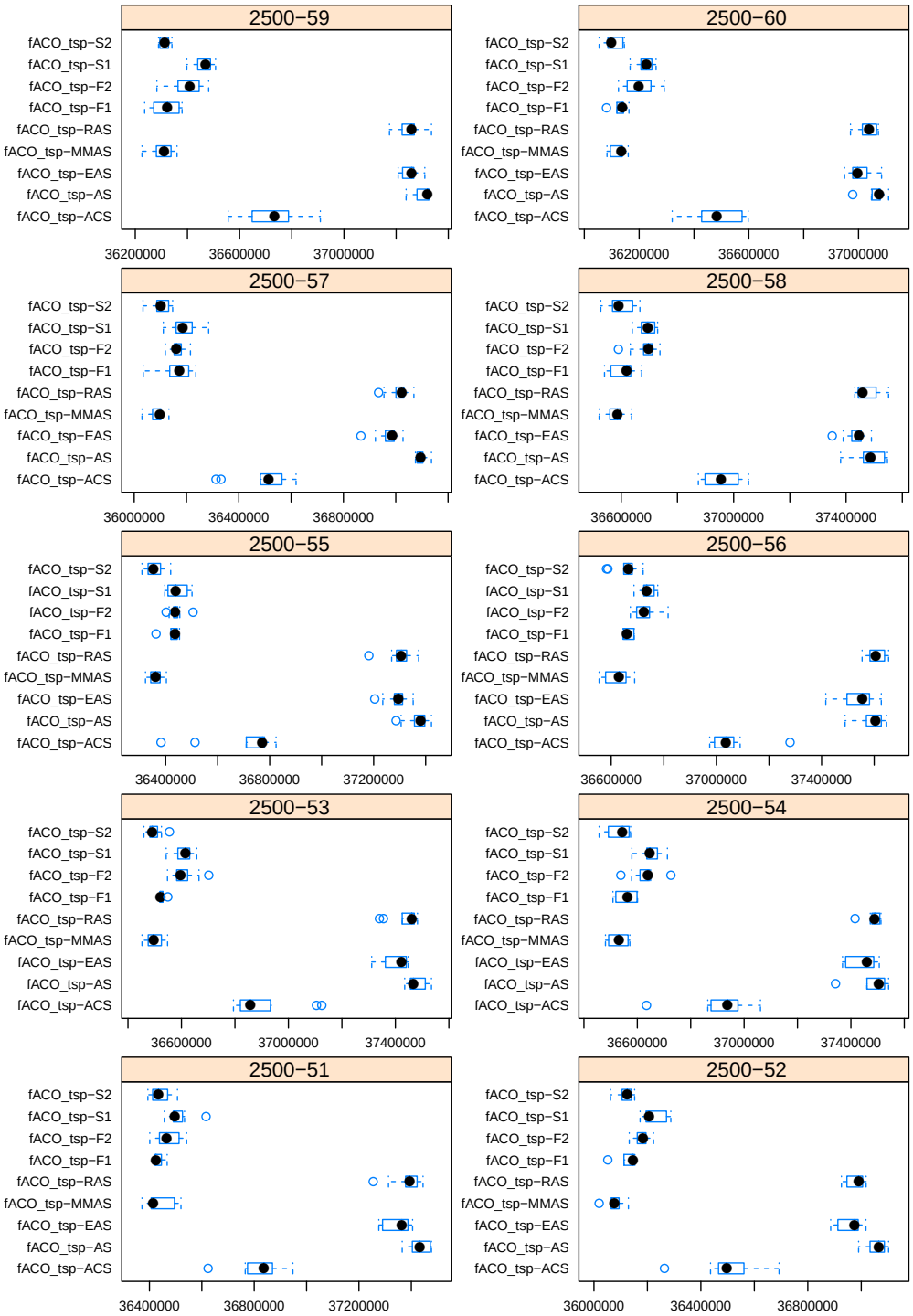

2000−59
2000−60
2000−57
2000−58
2000−55
2000−56
2000−53
2000−54
2000−51
2000−52
fACO_tsp−S2
fACO_tsp−S1
fACO_tsp−F2
fACO_tsp−F1
fACO_tsp−RAS
fACO_tsp−MMAS
fACO_tsp−EAS
fACO_tsp−AS
fACO_tsp−ACS
32600000 33000000 33400000
32400000 32800000 33200000
32600000 32800000 33000000 33200000 33400000
32800000 33000000 33200000 33400000 33600000
32600000 32800000 33000000 33200000 33400000 33600000
32600000 33000000 33400000
32800000 33000000 33200000 33400000 33600000
32800000 33000000 33200000 33400000 33600000
32600000 32800000 33000000 33200000 33400000
32200000 32400000 32600000 32800000 33000000

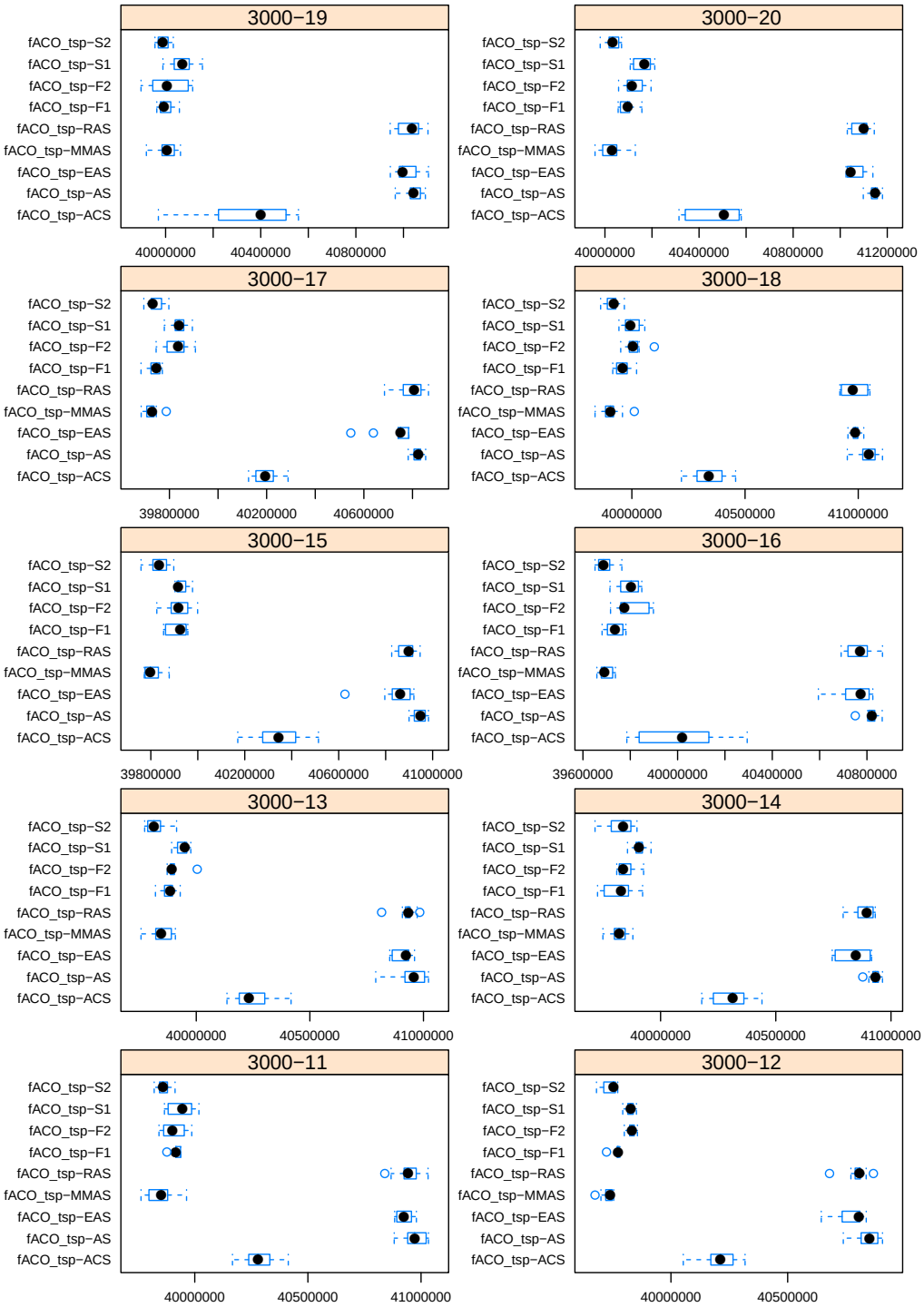

2500−19
2500−20
2500−17
2500−18
2500−15
2500−16
2500−13
2500−14
2500−11
2500−12
fACO_tsp−S2
fACO_tsp−S1
fACO_tsp−F2
fACO_tsp−F1
fACO_tsp−RAS
fACO_tsp−MMAS
fACO_tsp−EAS
fACO_tsp−AS
fACO_tsp−ACS
36400000
36800000
37200000
36600000
37000000
37400000
36200000
36600000
37000000
36400000
36800000
37200000
36600000
37000000
37400000
36400000
36800000
37200000
36200000
36600000
37000000
36400000
36800000
37200000
36400000
36800000
37200000
36200000
36600000
37000000

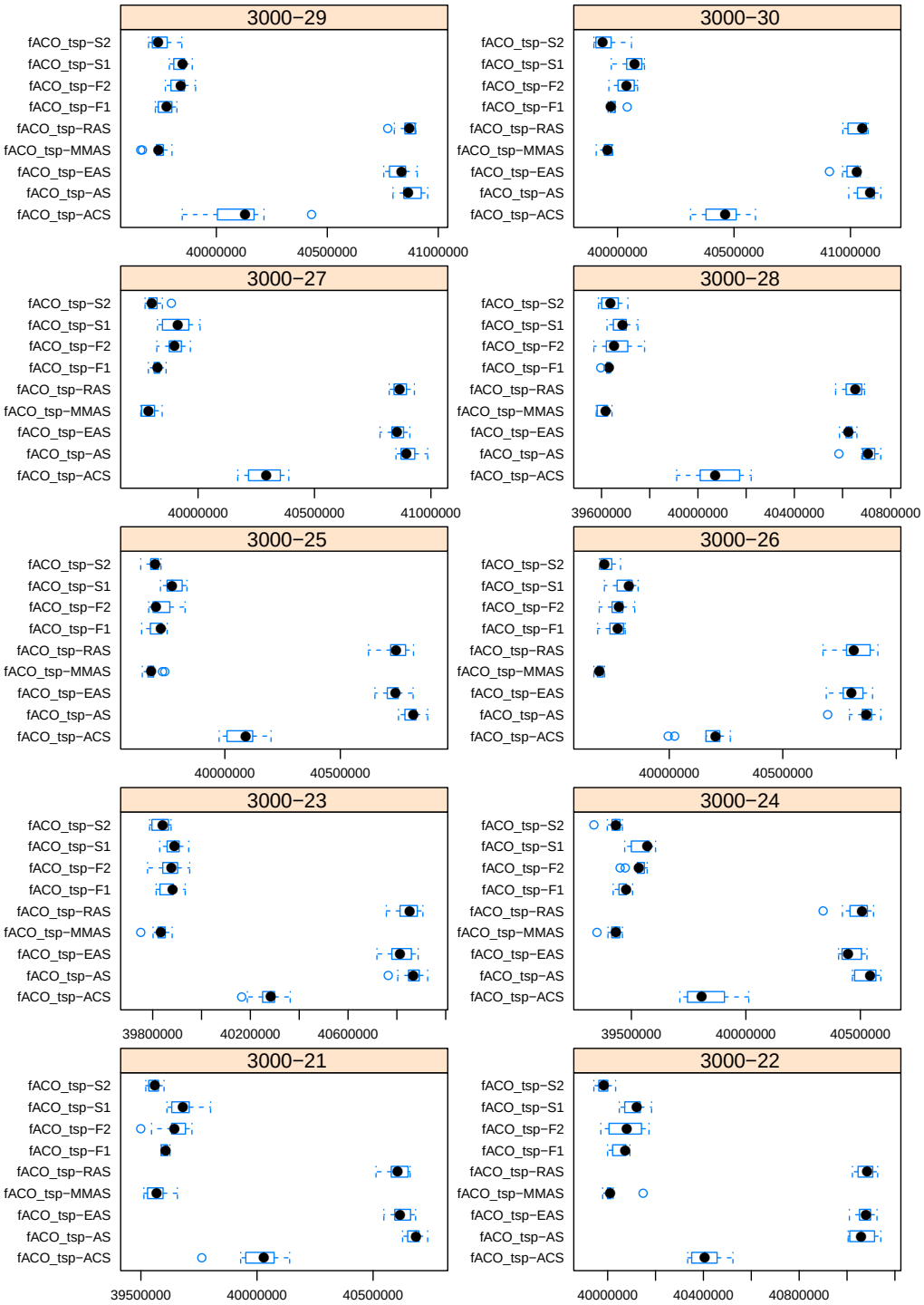

2500−29
2500−30
2500−27
2500−28
2500−25
2500−26
2500−23
2500−24
2500−21
2500−22
fACO_tsp−S2
fACO_tsp−S1
fACO_tsp−F2
fACO_tsp−F1
fACO_tsp−RAS
fACO_tsp−MMAS
fACO_tsp−EAS
fACO_tsp−AS
fACO_tsp−ACS
36400000
36800000
37200000
36600000
37000000
37400000
36200000

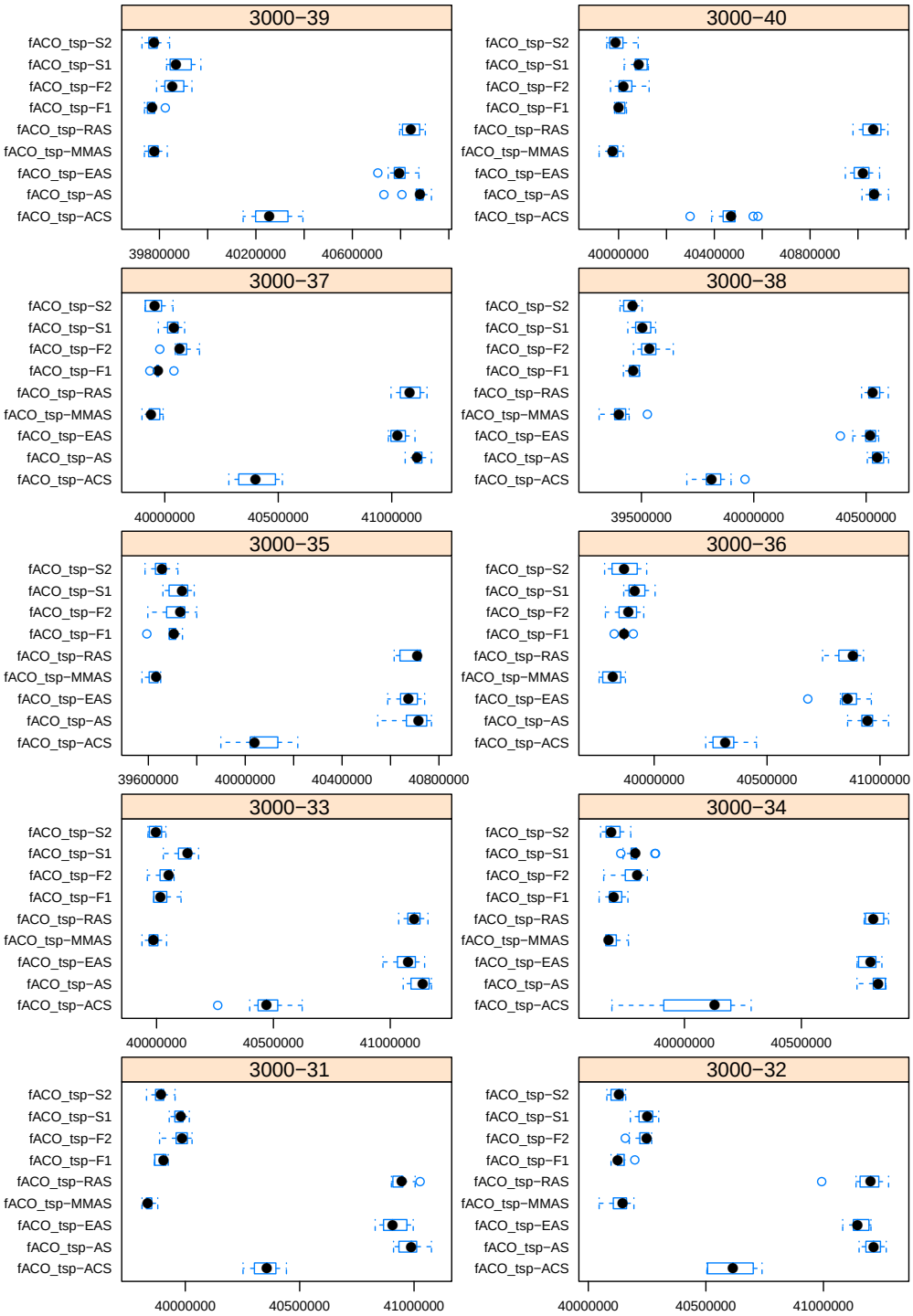

2500−39
fACO_tsp−S2
fACO_tsp−S1
fACO_tsp−F2
fACO_tsp−F1
fACO_tsp−RAS
fACO_tsp−MMAS
fACO_tsp−EAS
fACO_tsp−AS
fACO_tsp−ACS
36600000
37000000
37400000
2500−40
36400000
36800000
37200000
2500−37
36400000
36800000
37200000
37600000
2500−38
36000000
36400000
36800000
2500−35
36200000
36600000
37000000
2500−36
36400000
36800000
37200000
2500−33
36400000
36800000
37200000
2500−34
36400000
36800000
37200000
2500−31
36400000
36800000
37200000
2500−32
36600000
37000000
37400000

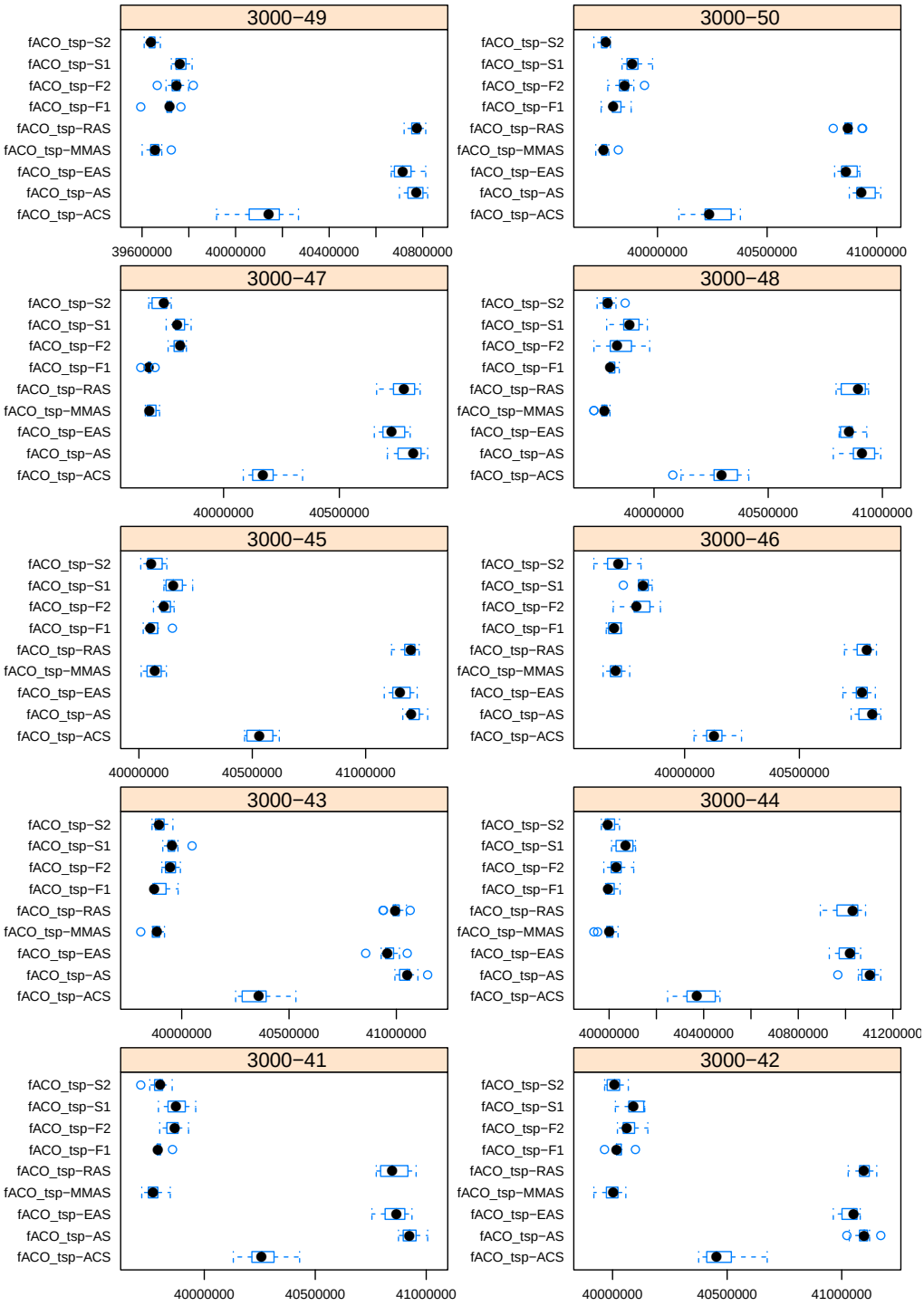

2500−49
fACO_tsp−S2
fACO_tsp−S1
fACO_tsp−F2
fACO_tsp−F1
fACO_tsp−RAS
fACO_tsp−MMAS
fACO_tsp−EAS
fACO_tsp−AS
fACO_tsp−ACS
36200000
36600000
37000000
2500−50
36600000
37000000
37400000
2500−47
36400000
36800000
37200000
2500−48
36400000
36800000
37200000
2500−45
36600000
37000000
37400000
2500−46
36200000
36600000
37000000
2500−43
36600000
37000000
37400000
2500−44
36400000
36800000
37200000
2500−41
36400000
36800000
37200000
2500−42
36600000
37000000
37400000

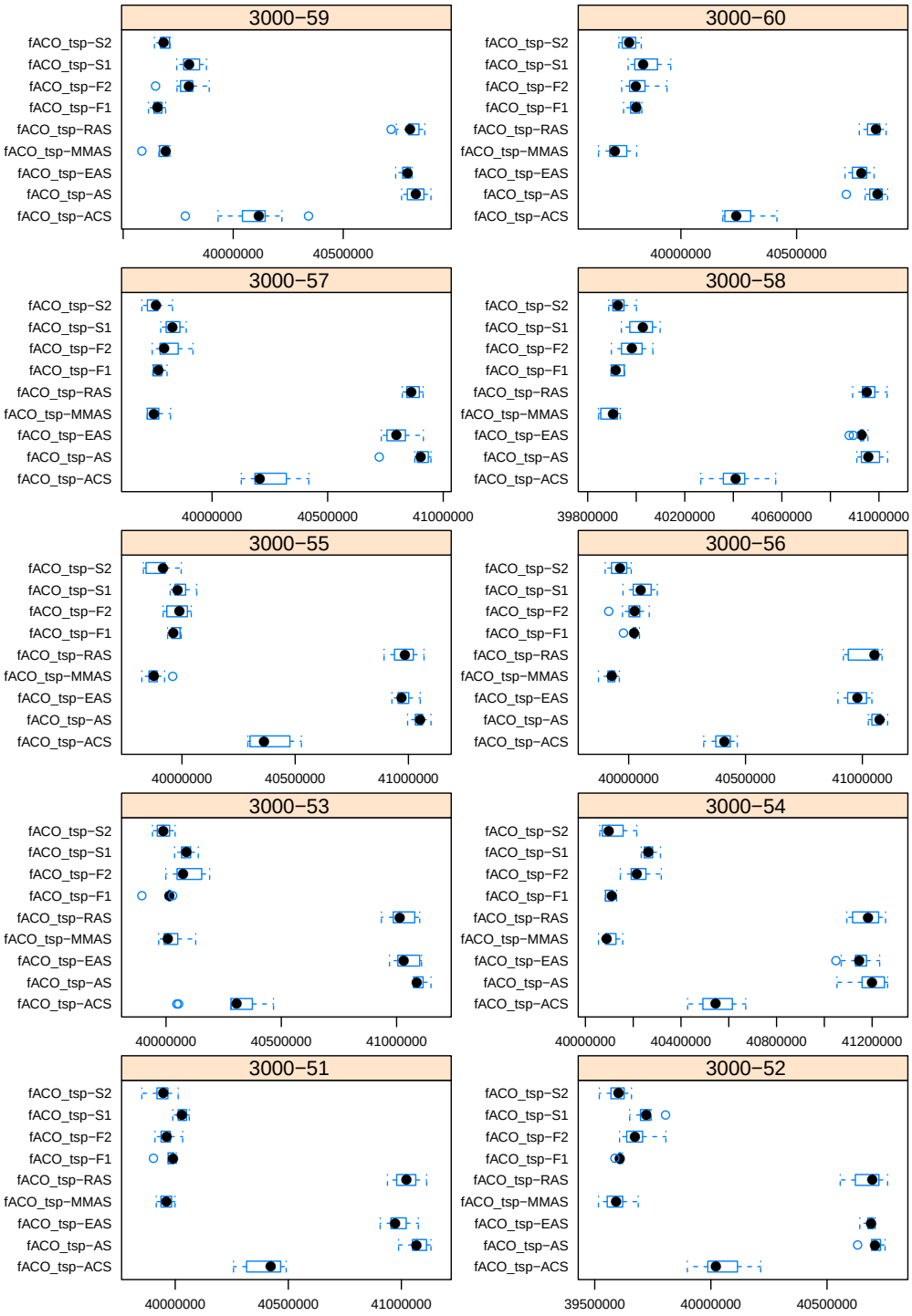

2500−59
2500−60
2500−57
2500−58
2500−55
2500−56
2500−53
2500−54
2500−51
2500−52
fACO_tsp−S2
fACO_tsp−S1
fACO_tsp−F2
fACO_tsp−F1
fACO_tsp−RAS
fACO_tsp−MMAS
fACO_tsp−EAS
fACO_tsp−AS
fACO_tsp−ACS
36200000
36600000
37000000
36000000
36400000
36800000
37200000
37400000


3000−19
3000−20
3000−17
3000−18
3000−15
3000−16
3000−13
3000−14
3000−11
3000−12
fACO_tsp−S2
fACO_tsp−S1
fACO_tsp−F2
fACO_tsp−F1
fACO_tsp−RAS
fACO_tsp−MMAS
fACO_tsp−EAS
fACO_tsp−AS
fACO_tsp−ACS
40000000
40400000
40800000
41200000
39800000
40200000
40600000
41000000
40500000
39600000
40400000
40800000

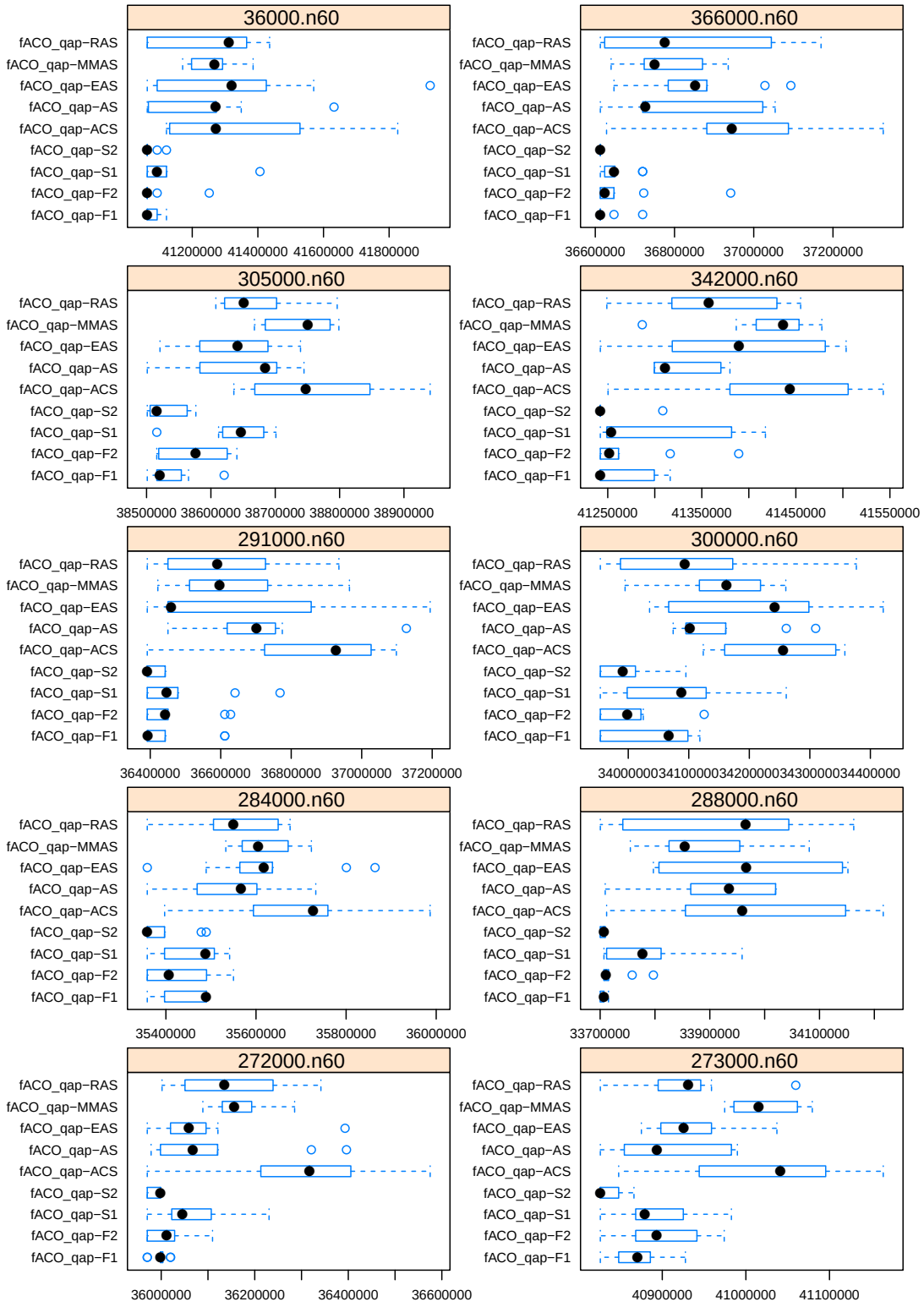

3000−29
3000−30
3000−27
3000−28
3000−25
3000−26
3000−23
3000−24
3000−21
3000−22
fACO_tsp−S2
fACO_tsp−S1
fACO_tsp−F2
fACO_tsp−F1
fACO_tsp−RAS
fACO_tsp−MMAS
fACO_tsp−EAS
fACO_tsp−AS
fACO_tsp−ACS
40000000
40500000
41000000
39600000
40400000
40800000
39800000
40200000
40600000
39500000

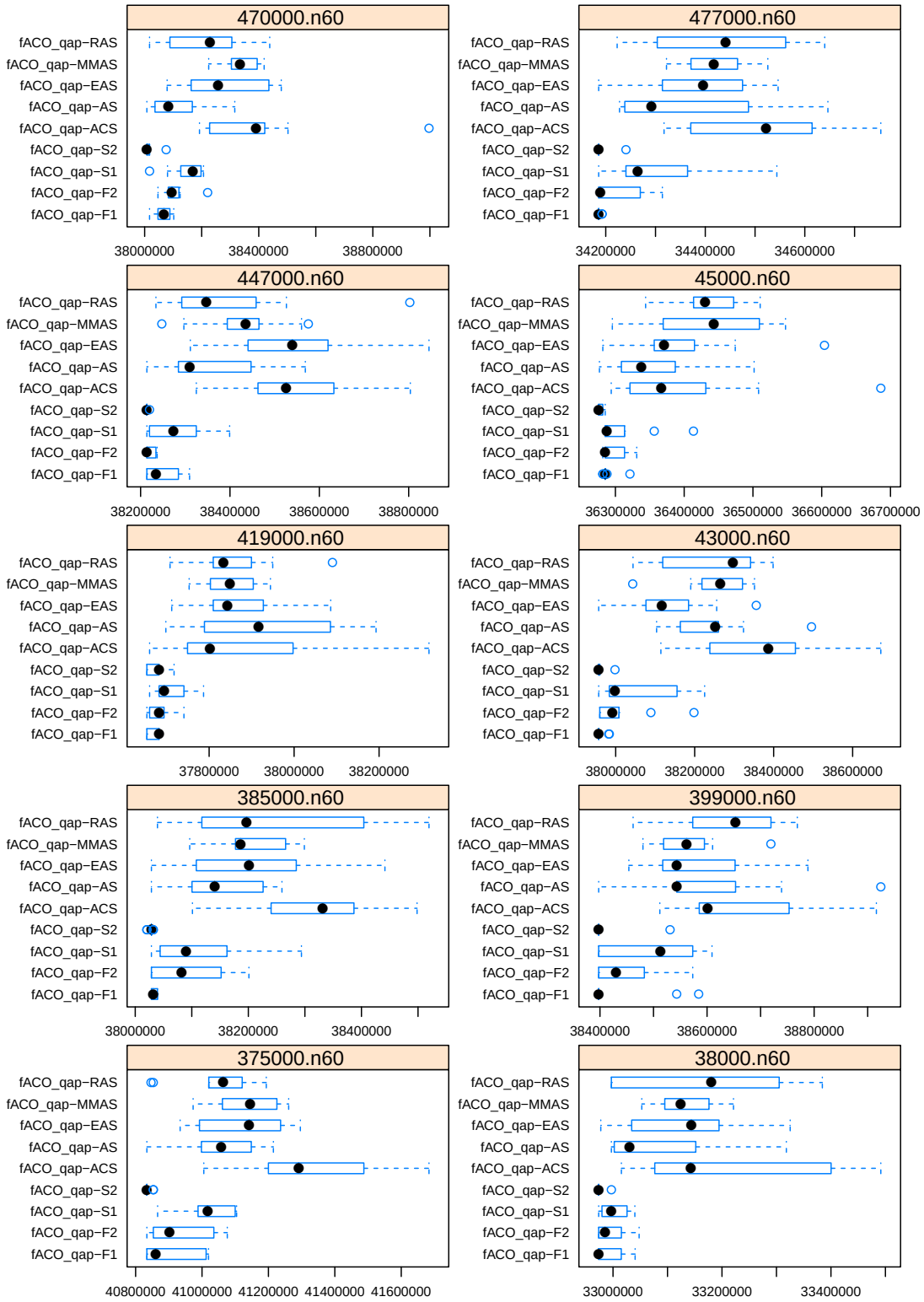

3000−39
3000−40
3000−37
3000−38
3000−35
3000−36
3000−33
3000−34
3000−31
3000−32
fACO_tsp−S2
fACO_tsp−S1
fACO_tsp−F2
fACO_tsp−F1
fACO_tsp−RAS
fACO_tsp−MMAS
fACO_tsp−EAS
fACO_tsp−AS
fACO_tsp−ACS
39800000
40200000
40600000
40000000
40400000
40800000
40000000
40500000
41000000
39500000
40000000
40500000
39600000
40000000
40400000
40800000
40000000
40500000
41000000
40000000
40500000
41000000
40000000
40500000
41000000

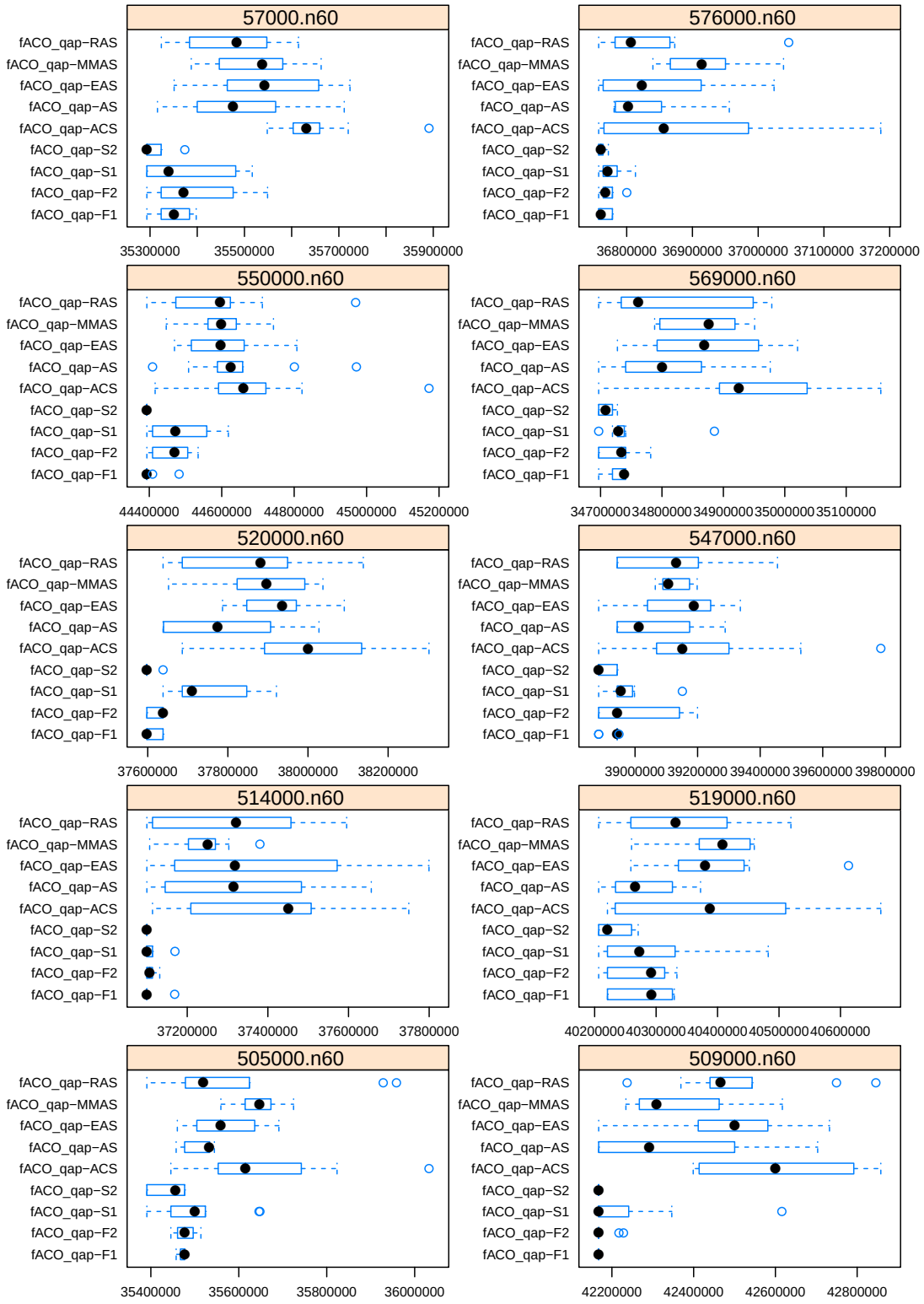

3000−49
3000−50
3000−47
3000−48
3000−45
3000−46
3000−43
3000−44
3000−41
3000−42
fACO_tsp−S2
fACO_tsp−S1
fACO_tsp−F2
fACO_tsp−F1
fACO_tsp−RAS
fACO_tsp−MMAS
fACO_tsp−EAS
fACO_tsp−AS
fACO_tsp−ACS
39600000
40000000
40400000
40800000
40000000
40500000
41000000
40000000
40500000
40000000
40500000
41000000
40000000
40500000
40000000
40400000
40800000
41200000
40000000
40500000
41000000
40000000
40500000
41000000

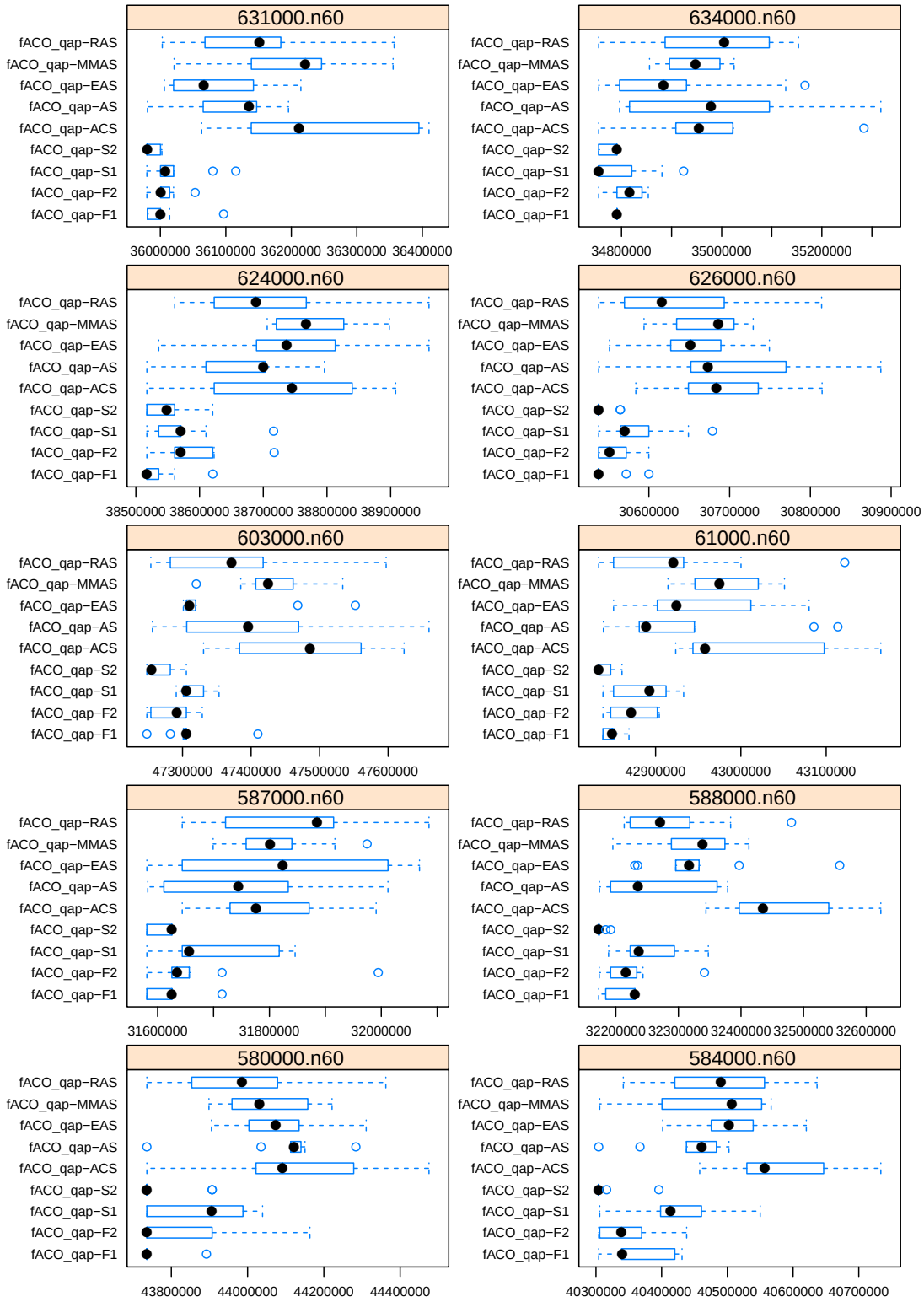

3000−59
3000−60
3000−57
3000−58
3000−55
3000−56
3000−53
3000−54
3000−51
3000−52
fACO_tsp−S2
fACO_tsp−S1
fACO_tsp−F2
fACO_tsp−F1
fACO_tsp−RAS
fACO_tsp−MMAS
fACO_tsp−EAS
fACO_tsp−AS
fACO_tsp−ACS
40000000
40500000
41000000
39800000
40200000
40600000
40400000
40800000
41200000
39500000

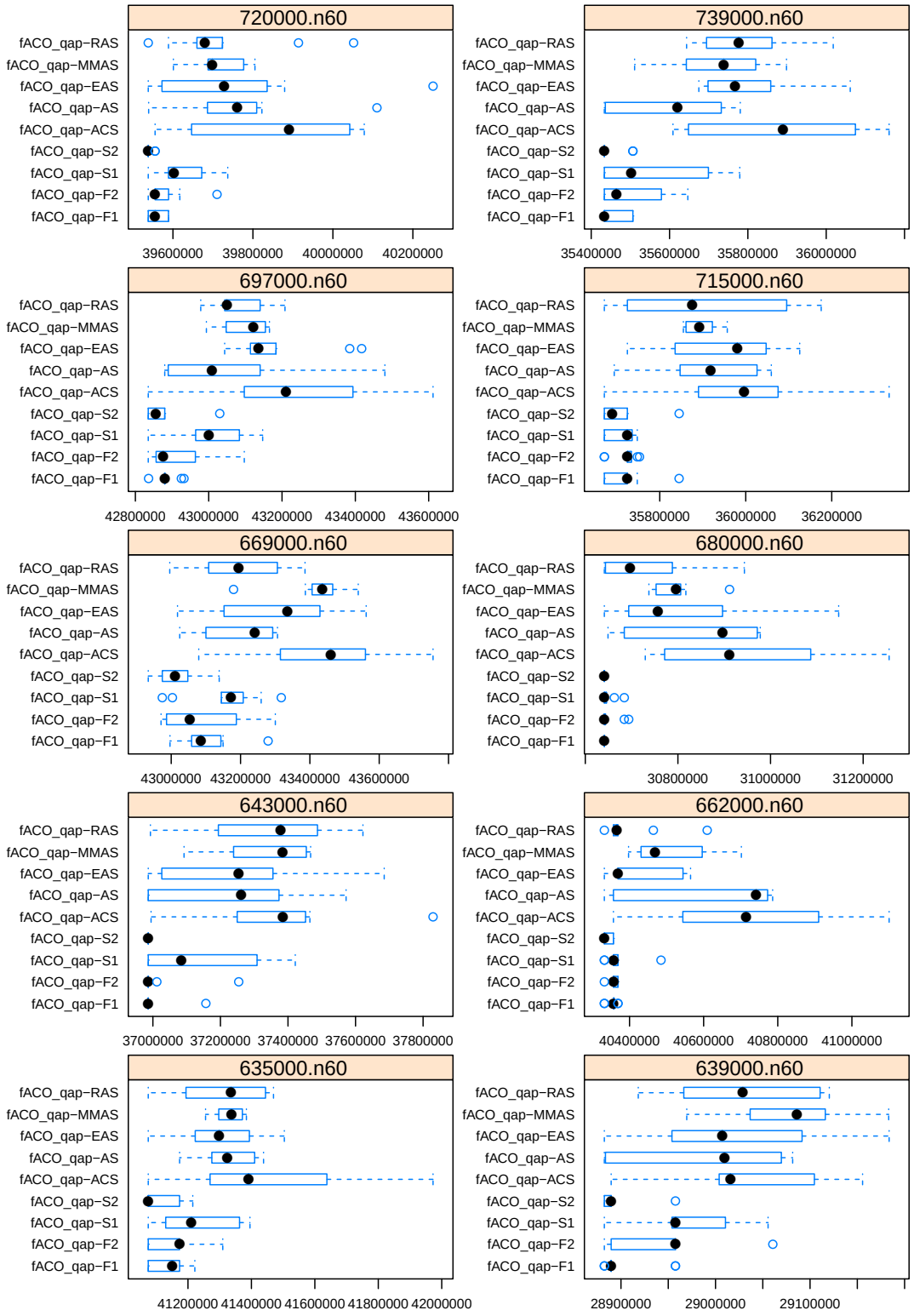
Quadratic Assignment Problem

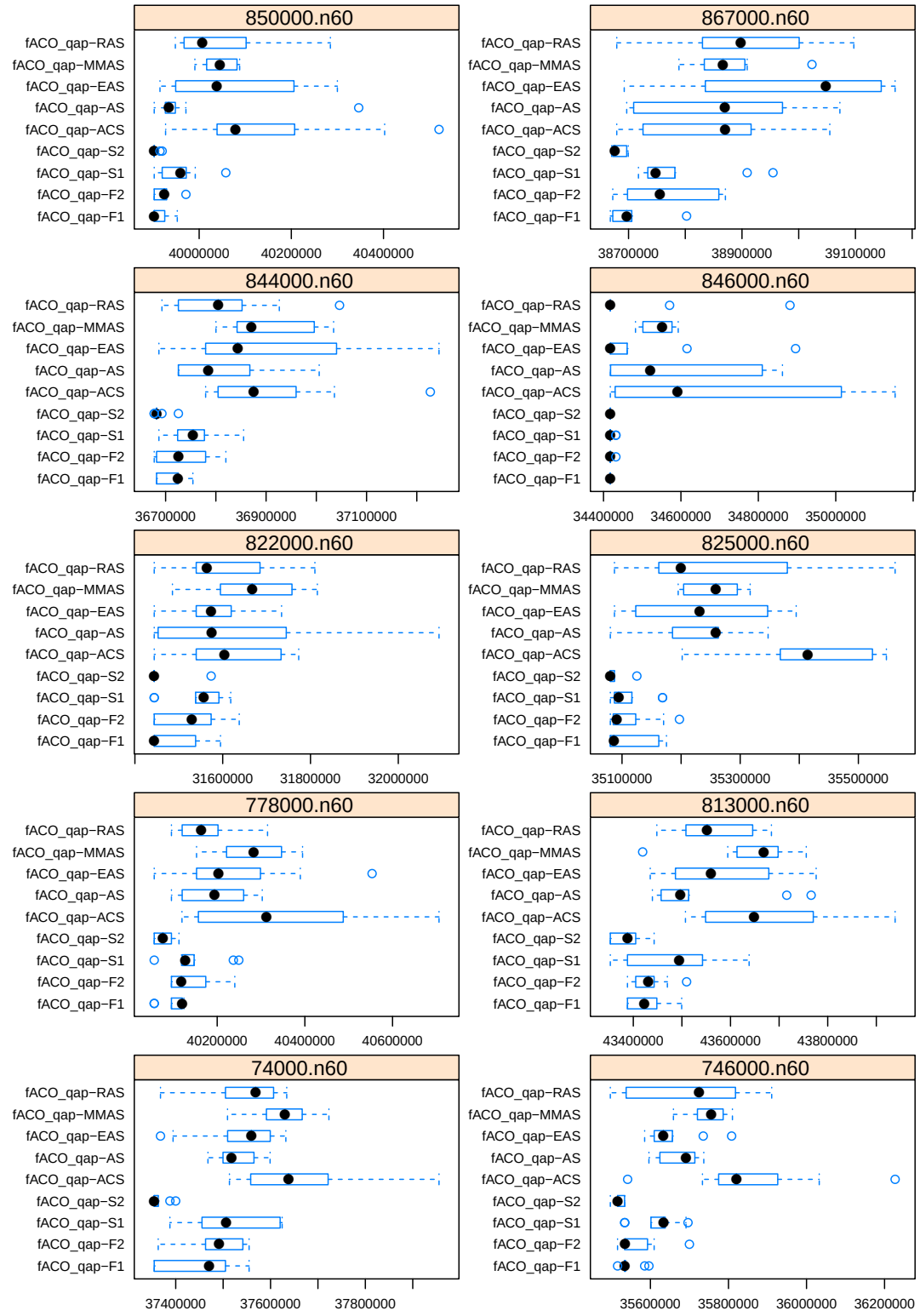

36000.n60
366000.n60
305000.n60
342000.n60
291000.n60
300000.n60
284000.n60
288000.n60
272000.n60
273000.n60
fACO_qap−RAS
fACO_qap−MMAS
fACO_qap−EAS
fACO_qap−AS
fACO_qap−ACS
fACO_qap−S2
fACO_qap−S1
fACO_qap−F2
fACO_qap−F1
41200000 41400000 41600000 41800000
36600000 36800000 37000000 37200000
38500000 38600000 38700000 38800000 38900000
41250000 41350000 41450000 41550000
36400000 36600000 36800000 37000000 37200000
34000000 34100000 34200000 34300000 34400000
35400000 35600000 35800000 36000000
33700000 33900000 34100000
36000000 36200000 36400000 36600000
40900000 41000000 41100000

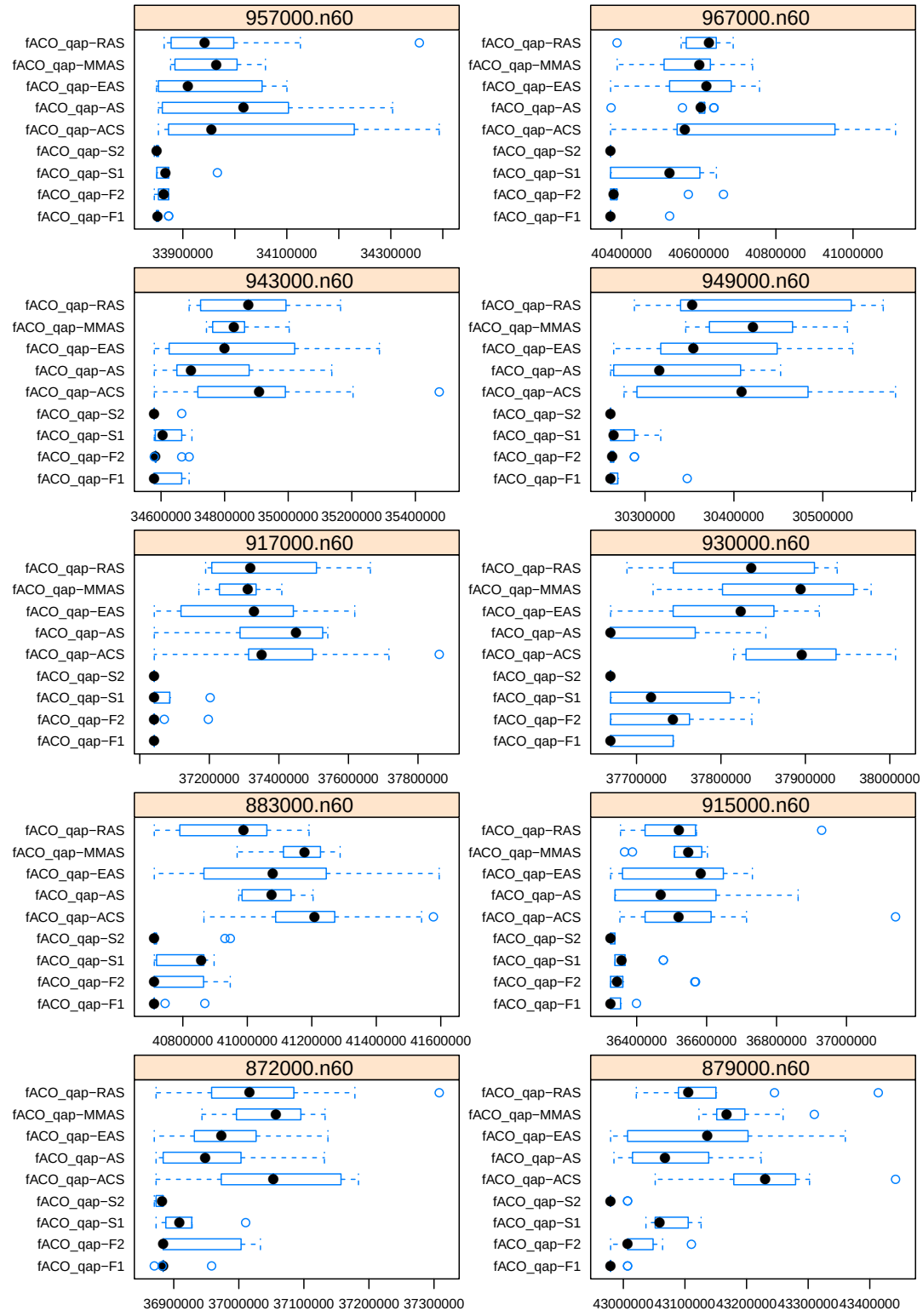

470000.n60
fACO_qap−RAS
fACO_qap−MMAS
fACO_qap−EAS
fACO_qap−AS
fACO_qap−ACS
fACO_qap−S2
fACO_qap−S1
fACO_qap−F2
fACO_qap−F1
38000000
38400000
38800000
477000.n60
34200000
34400000
34600000
447000.n60
38200000
38400000
38600000
38800000
45000.n60
36300000
36400000
36500000
36600000
36700000
419000.n60
37800000
38000000
38200000
43000.n60
38000000
38200000
38400000
38600000
385000.n60
38000000
38200000
38400000
399000.n60
38400000
38600000
38800000
375000.n60
40800000
41000000
41200000
41400000
41600000
38000.n60
33000000
33200000
33400000

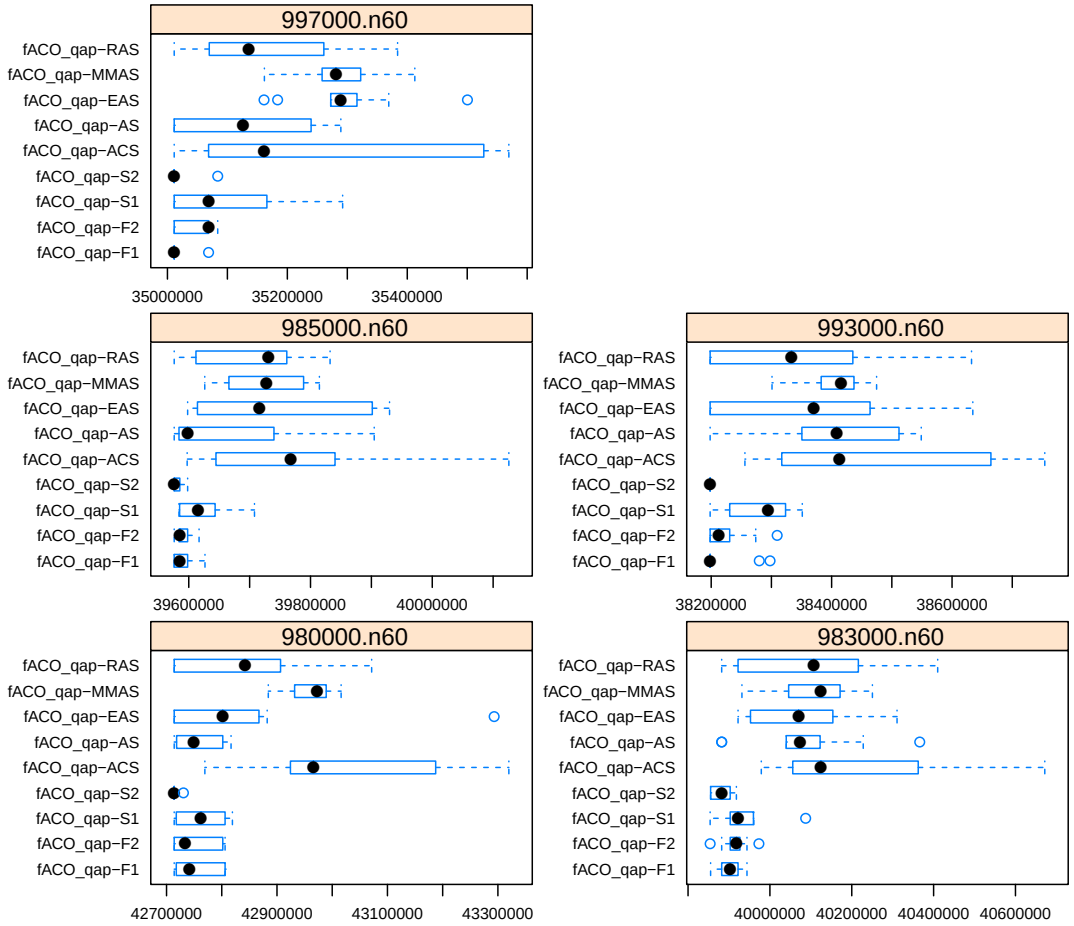

57000.n60
576000.n60
550000.n60
569000.n60
520000.n60
547000.n60
514000.n60
519000.n60
505000.n60
509000.n60
fACO_qap−RAS
fACO_qap−MMAS
fACO_qap−EAS
fACO_qap−AS
fACO_qap−ACS
fACO_qap−S2
fACO_qap−S1
fACO_qap−F2
fACO_qap−F1
35300000 35500000 35700000 35900000
36800000 36900000 37000000 37100000 37200000
44400000 44600000 44800000 45000000 45200000
34700000 34800000 34900000 35000000 35100000
37600000 37800000 38000000 38200000
39000000 39200000 39400000 39600000 39800000
37200000 37400000 37600000 37800000
40200000 40300000 40400000 40500000 40600000
35400000 35600000 35800000 36000000
42200000 42400000 42600000 42800000

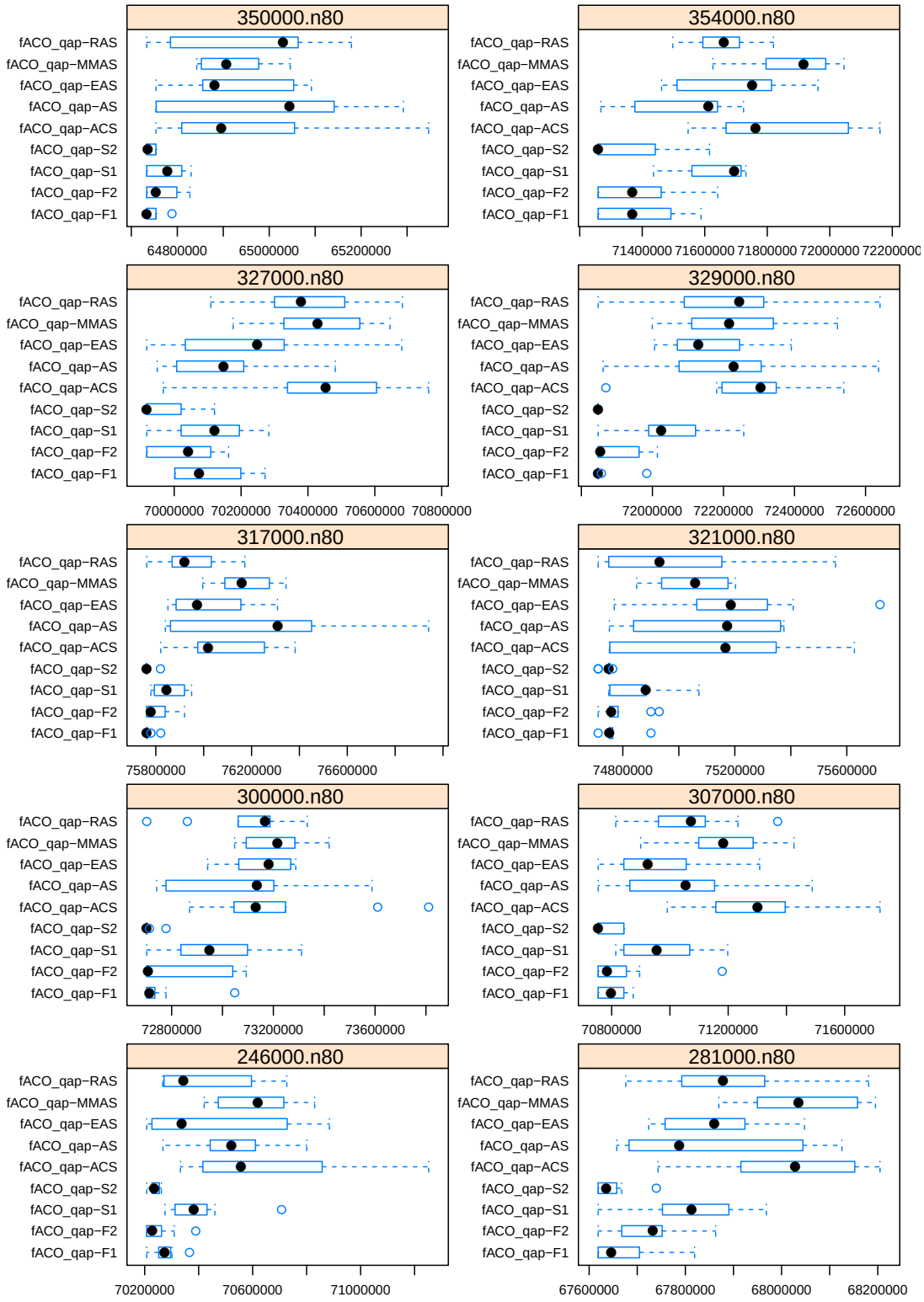

631000.n60
fACO_qap−RAS
fACO_qap−MMAS
fACO_qap−EAS
fACO_qap−AS
fACO_qap−ACS
fACO_qap−S2
fACO_qap−S1
fACO_qap−F2
fACO_qap−F1
36000000 36100000 36200000 36300000 36400000
634000.n60
34800000 35000000 35200000
624000.n60
38500000 38600000 38700000 38800000 38900000
626000.n60
30600000 30700000 30800000 30900000
603000.n60
47300000 47400000 47500000 47600000
61000.n60
42900000 43000000 43100000
587000.n60
31600000 31800000 32000000
588000.n60
32200000 32300000 32400000 32500000 32600000
580000.n60
43800000 44000000 44200000 44400000
584000.n60
40300000 40400000 40500000 40600000 40700000

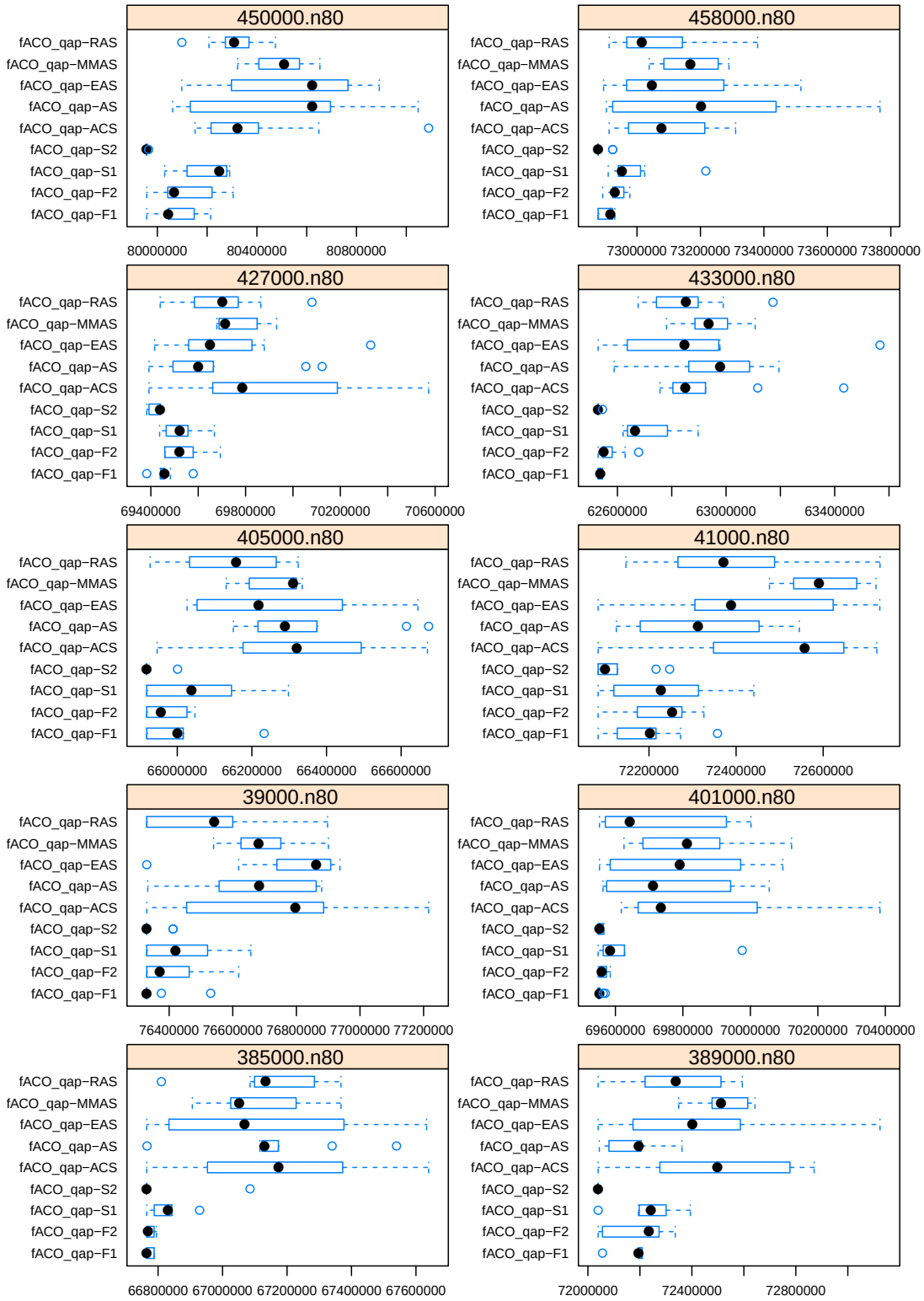

720000.n60
fACO_qap−RAS
fACO_qap−MMAS
fACO_qap−EAS
fACO_qap−AS
fACO_qap−ACS
fACO_qap−S2
fACO_qap−S1
fACO_qap−F2
fACO_qap−F1
39600000 39800000 40000000 40200000
739000.n60
35400000 35600000 35800000 36000000
697000.n60
42800000 43000000 43200000 43400000 43600000
715000.n60
35800000 36000000 36200000
669000.n60
43000000 43200000 43400000 43600000
680000.n60
30800000 31000000 31200000
643000.n60
37000000 37200000 37400000 37600000 37800000
662000.n60
40400000 40600000 40800000 41000000
635000.n60
41200000 41400000 41600000 41800000 42000000
639000.n60
28900000 29000000 29100000

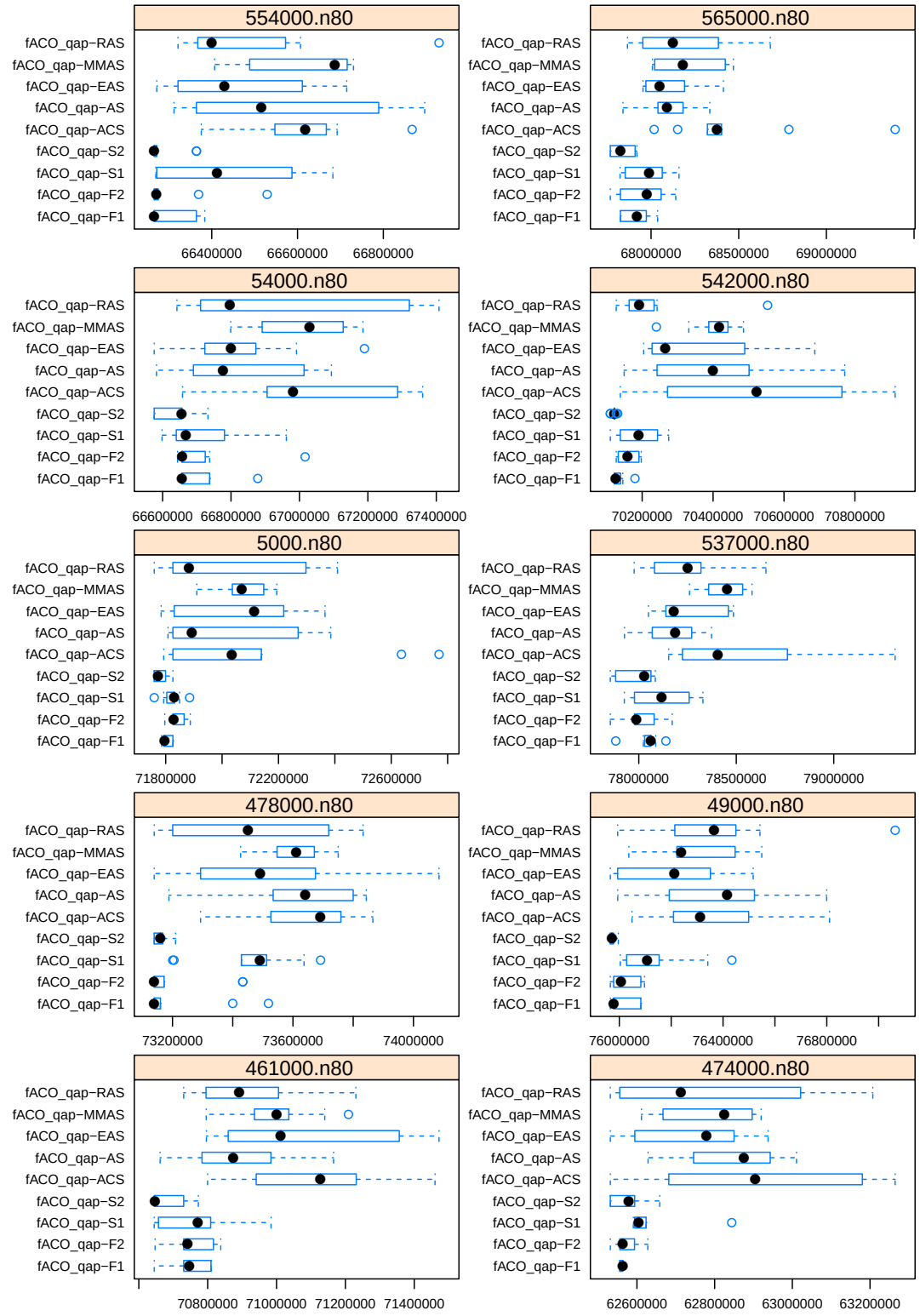

850000.n60
867000.n60
844000.n60
846000.n60
822000.n60
825000.n60
778000.n60
813000.n60
74000.n60
746000.n60
fACO_qap−RAS
fACO_qap−MMAS
fACO_qap−EAS
fACO_qap−AS
fACO_qap−ACS
fACO_qap−S2
fACO_qap−S1
fACO_qap−F2
fACO_qap−F1
40000000
40200000
40400000
38700000
38900000
39100000
36700000
36900000
37100000
34400000
34600000
34800000
35000000
31600000
31800000
32000000
35100000
35300000
35500000
40200000
40400000
40600000
43400000
43600000
43800000
37400000
37600000
37800000
35600000
35800000
36000000
36200000

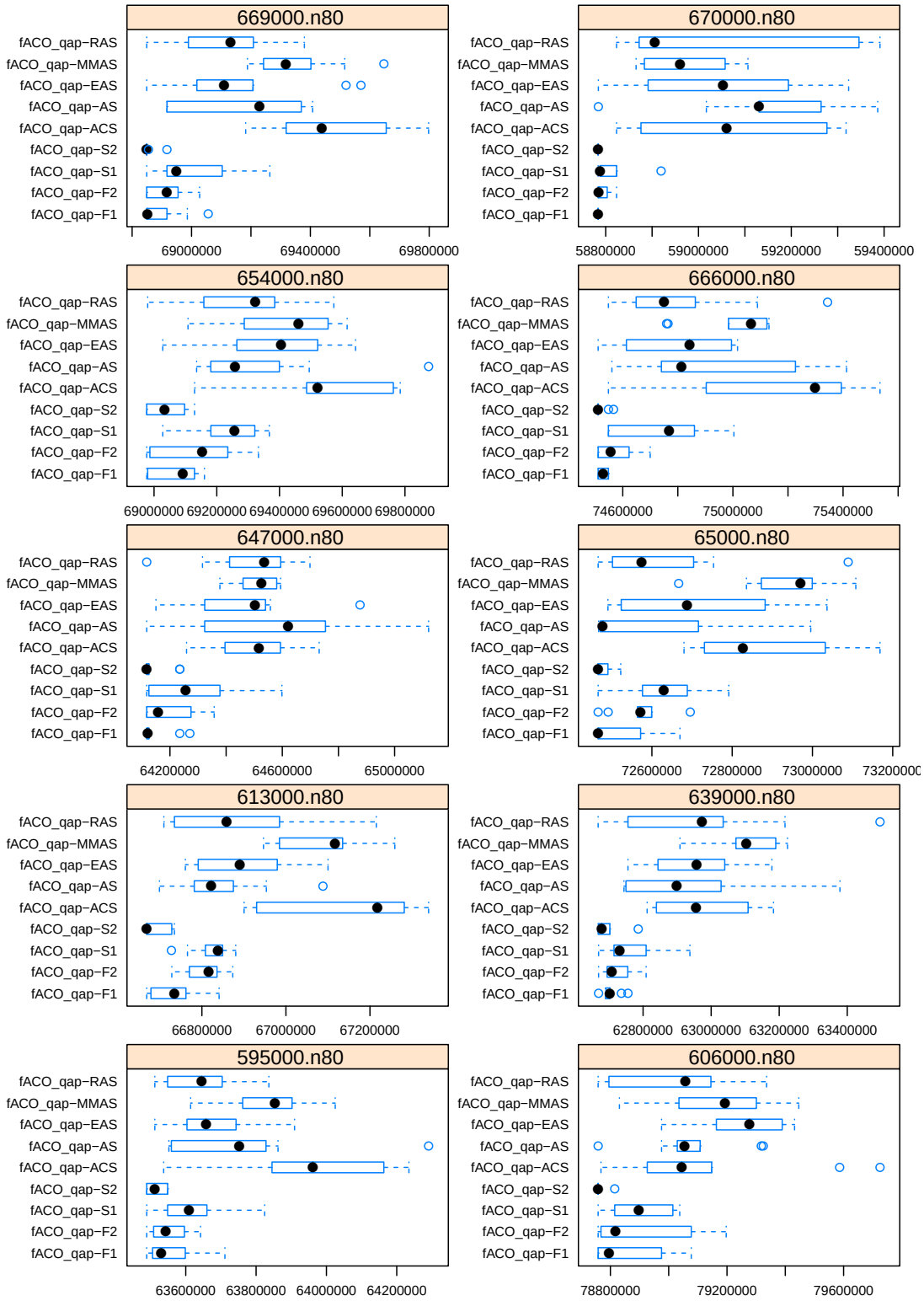

957000.n60
967000.n60
943000.n60
949000.n60
917000.n60
930000.n60
883000.n60
915000.n60
872000.n60
879000.n60
fACO_qap−RAS
fACO_qap−MMAS
fACO_qap−EAS
fACO_qap−AS
fACO_qap−ACS
fACO_qap−S2
fACO_qap−S1
fACO_qap−F2
fACO_qap−F1
33900000 34100000 34300000
40400000 40600000 40800000 41000000
34600000 34800000 35000000 35200000 35400000
30300000 30400000 30500000
37200000 37400000 37600000 37800000
37700000 37800000 37900000 38000000
40800000 41000000 41200000 41400000 41600000
36400000 36600000 36800000 37000000
36900000 37000000 37100000 37200000 37300000
43000000 43100000 43200000 43300000 43400000

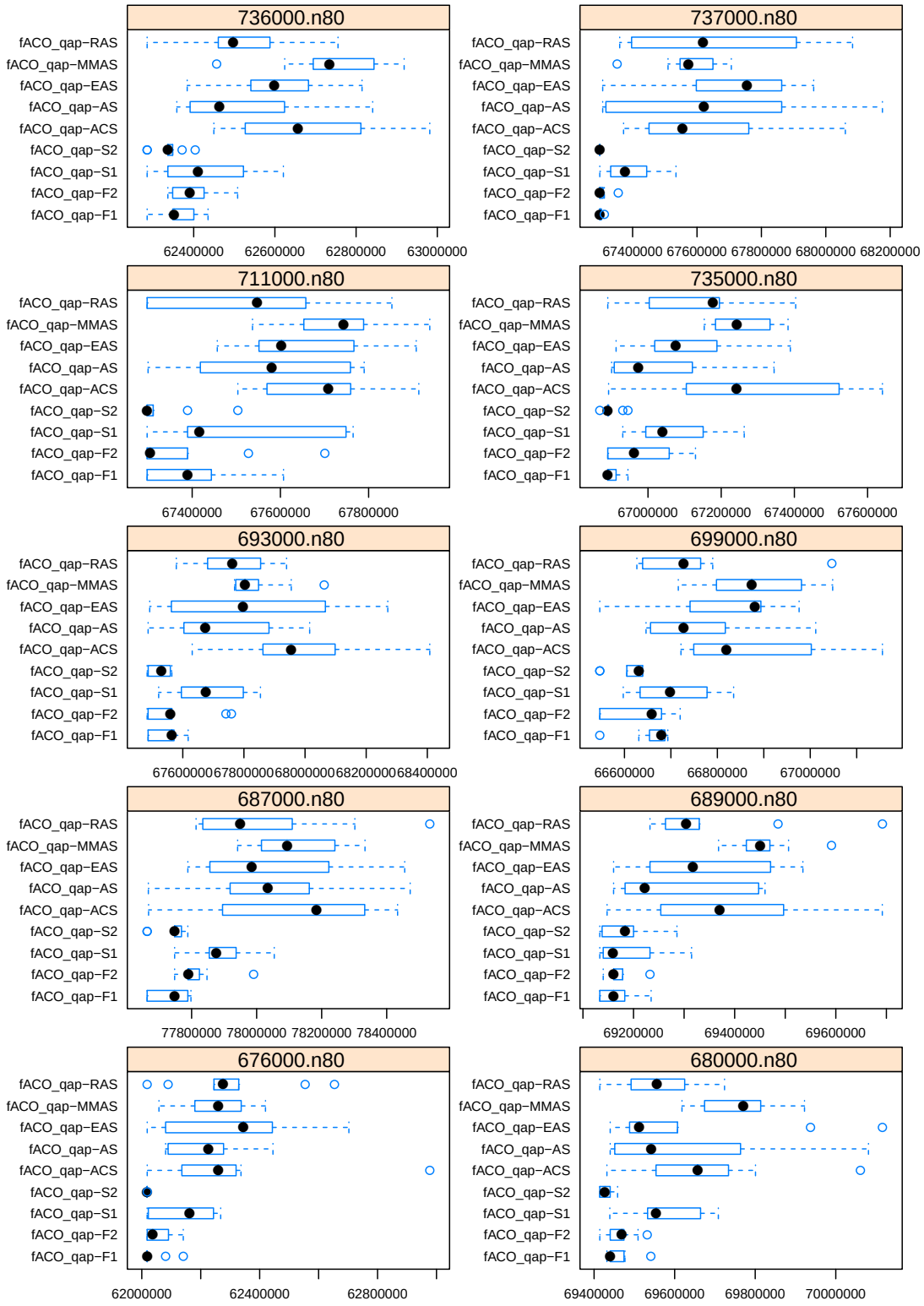

997000.n60
fACO_qap−RAS
fACO_qap−MMAS
fACO_qap−EAS
fACO_qap−AS
fACO_qap−ACS
fACO_qap−S2
fACO_qap−S1
fACO_qap−F2
fACO_qap−F1
35000000
35200000
35400000
985000.n60
39600000
39800000
40000000
993000.n60
38200000
38400000
38600000
980000.n60
42700000
42900000
43100000
43300000
983000.n60
40000000
40200000
40400000
40600000

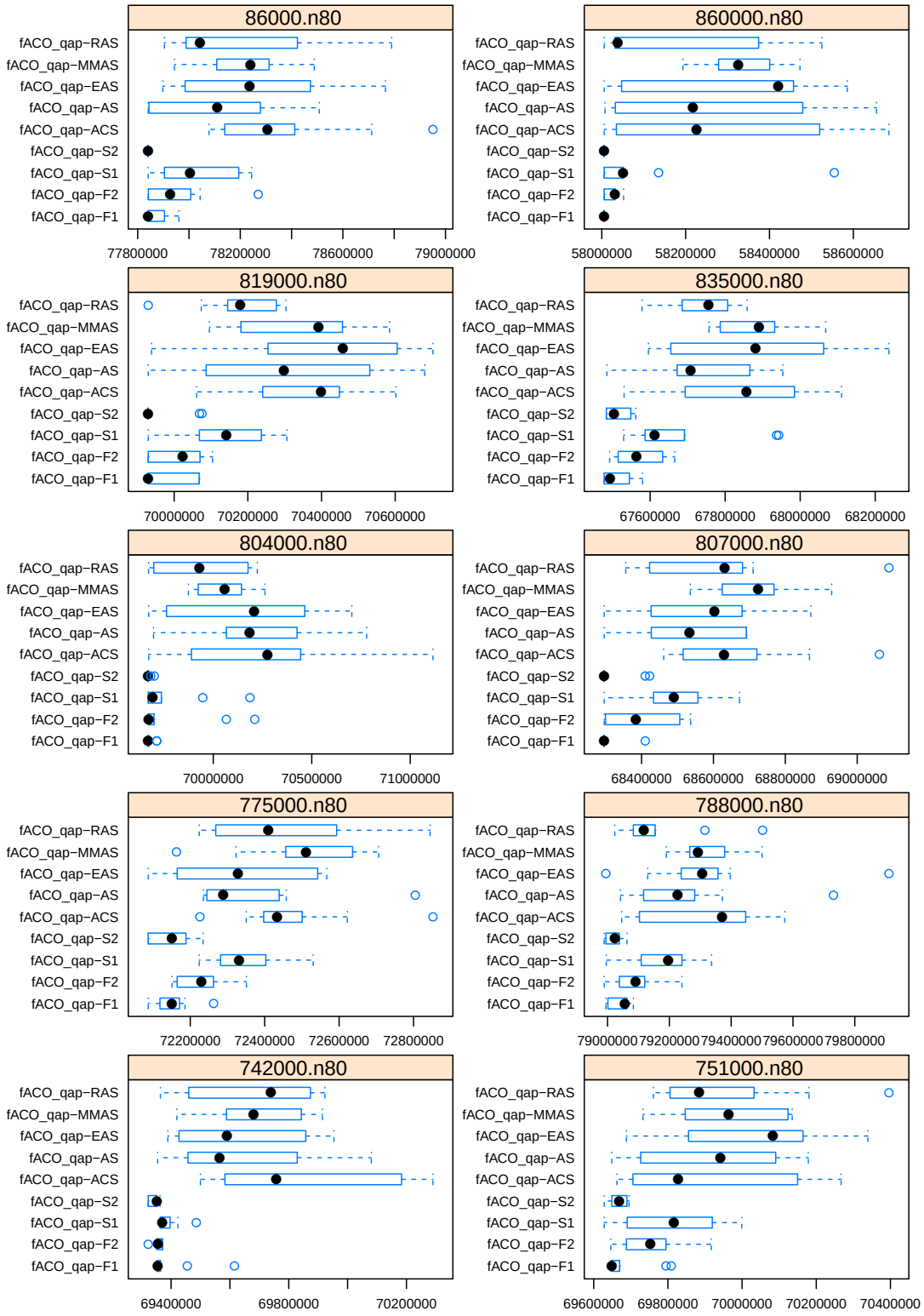

350000.n80
fACO_qap−RAS
fACO_qap−MMAS
fACO_qap−EAS
fACO_qap−AS
fACO_qap−ACS
fACO_qap−S2
fACO_qap−S1
fACO_qap−F2
fACO_qap−F1
64800000
65000000
65200000
354000.n80
71400000
71600000
71800000
72000000
72200000
327000.n80
70000000
70200000
70400000
70600000
70800000
329000.n80
72000000
72200000
72400000
72600000
317000.n80
75800000
76200000
76600000
321000.n80
74800000
75200000
75600000
300000.n80
72800000
73200000
73600000
307000.n80
70800000
71200000
71600000
246000.n80
70200000
70600000
71000000
281000.n80
67600000
67800000
68000000
68200000

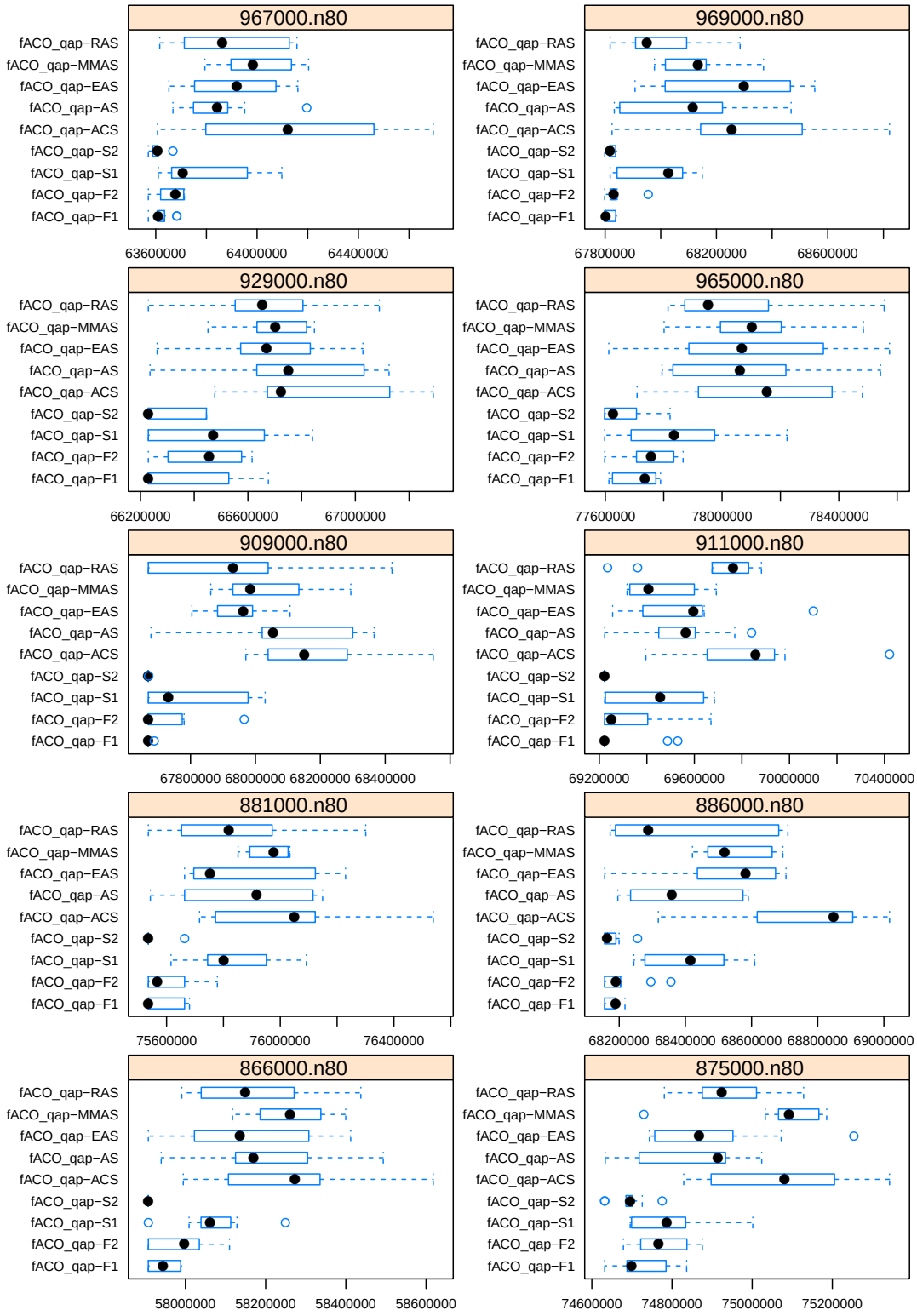

450000.n80
458000.n80
427000.n80
433000.n80
405000.n80
41000.n80
39000.n80
401000.n80
385000.n80
389000.n80
fACO_qap−RAS
fACO_qap−MMAS
fACO_qap−EAS
fACO_qap−AS
fACO_qap−ACS
fACO_qap−S2
fACO_qap−S1
fACO_qap−F2
fACO_qap−F1
80000000 80400000 80800000
73000000 73200000 73400000 73600000 73800000
69400000 69800000 70200000 70600000
62600000 63000000 63400000
66000000 66200000 66400000 66600000
72200000 72400000 72600000
76400000 76600000 76800000 77000000 77200000
69600000 69800000 70000000 70200000 70400000
66800000 67000000 67200000 67400000 67600000
72000000 72400000 72800000

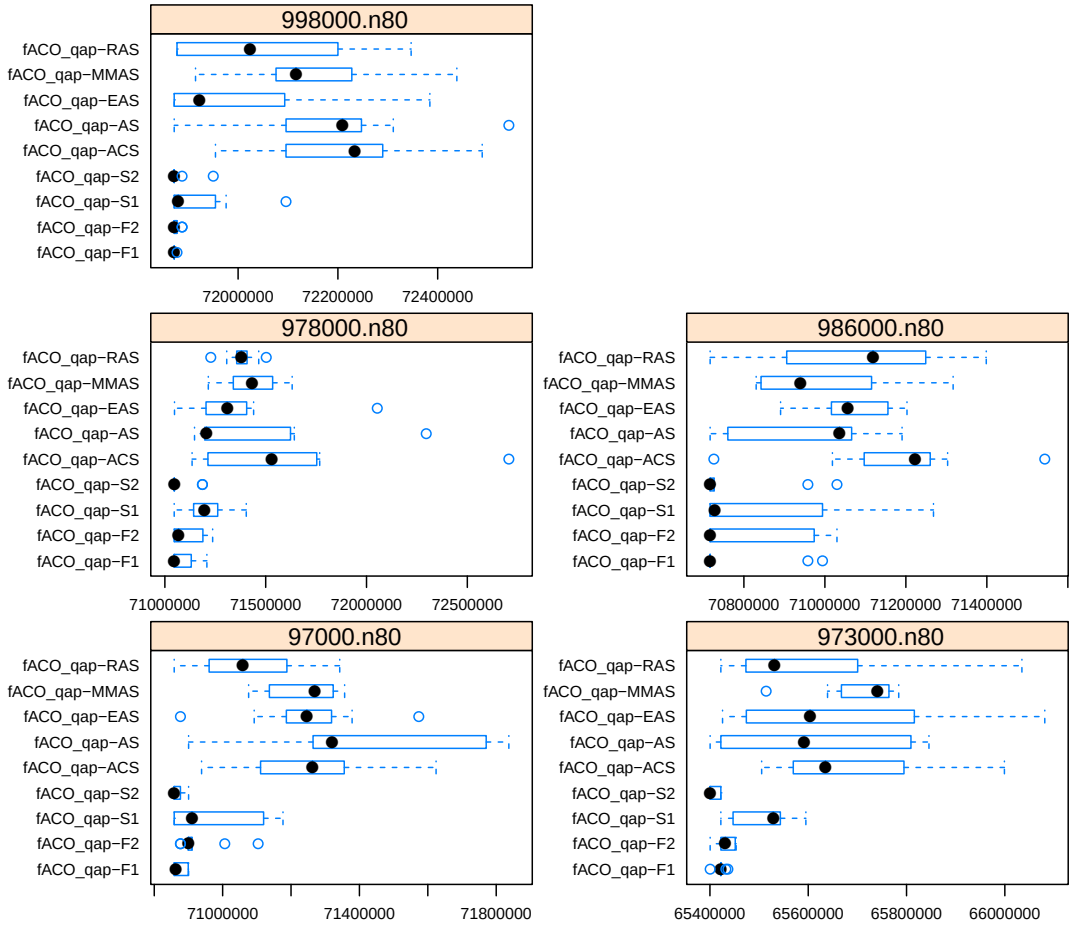

554000.n80
fACO_qap−RAS
fACO_qap−MMAS
fACO_qap−EAS
fACO_qap−AS
fACO_qap−ACS
fACO_qap−S2
fACO_qap−S1
fACO_qap−F2
fACO_qap−F1
66400000 66600000 66800000
565000.n80
68000000 68500000 69000000
54000.n80
66600000 66800000 67000000 67200000 67400000
542000.n80
70200000 70400000 70600000 70800000
5000.n80
71800000 72200000 72600000
537000.n80
78000000 78500000 79000000
478000.n80
73200000 73600000 74000000
49000.n80
76000000 76400000 76800000
461000.n80
70800000 71000000 71200000 71400000
474000.n80
62600000 62800000 63000000 63200000

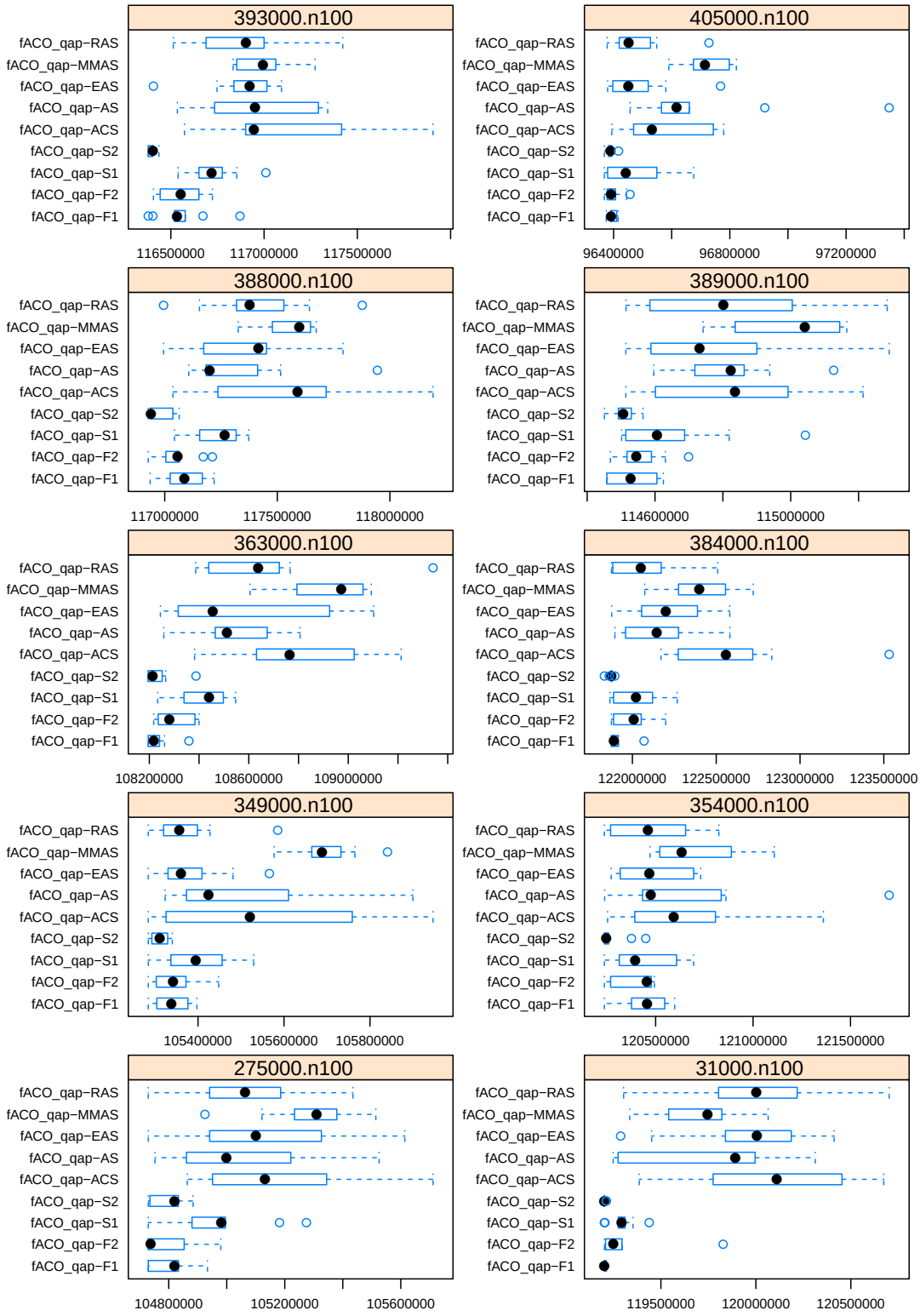

669000.n80
fACO_qap−RAS
fACO_qap−MMAS
fACO_qap−EAS
fACO_qap−AS
fACO_qap−ACS
fACO_qap−S2
fACO_qap−S1
fACO_qap−F2
fACO_qap−F1
69000000
69400000
69800000
670000.n80
58800000
59000000
59200000
59400000
654000.n80
69000000 69200000 69400000 69600000 69800000
666000.n80
74600000
75000000
75400000
647000.n80
64200000
64600000
65000000
65000.n80
72600000
72800000
73000000
613000.n80
66800000
67000000
67200000
639000.n80
62800000
63000000
63200000
63400000
595000.n80
63600000
63800000
64000000
64200000
606000.n80
78800000
79200000
79600000

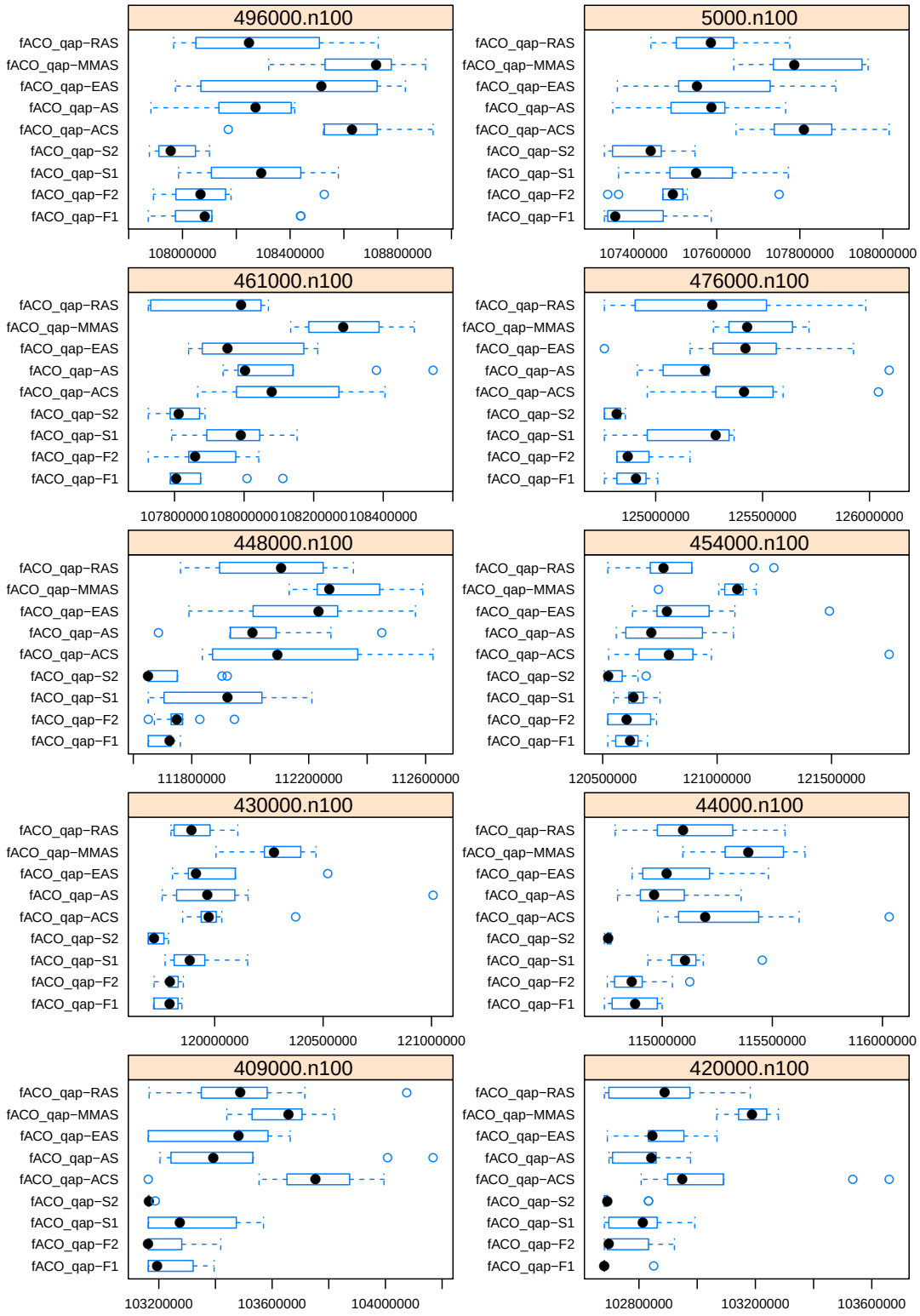

736000.n80
737000.n80
711000.n80
735000.n80
693000.n80
699000.n80
687000.n80
689000.n80
676000.n80
680000.n80
fACO_qap−RAS
fACO_qap−MMAS
fACO_qap−EAS
fACO_qap−AS
fACO_qap−ACS
fACO_qap−S2
fACO_qap−S1
fACO_qap−F2
fACO_qap−F1
62400000 62600000 62800000 63000000
67400000 67600000 67800000 68000000 68200000
67400000 67600000 67800000
67000000 67200000 67400000 67600000
67600000 67800000 68000000 68200000 68400000
66600000 66800000 67000000
77800000 78000000 78200000 78400000
69200000 69400000 69600000
62000000 62400000 62800000
69400000 69600000 69800000 70000000

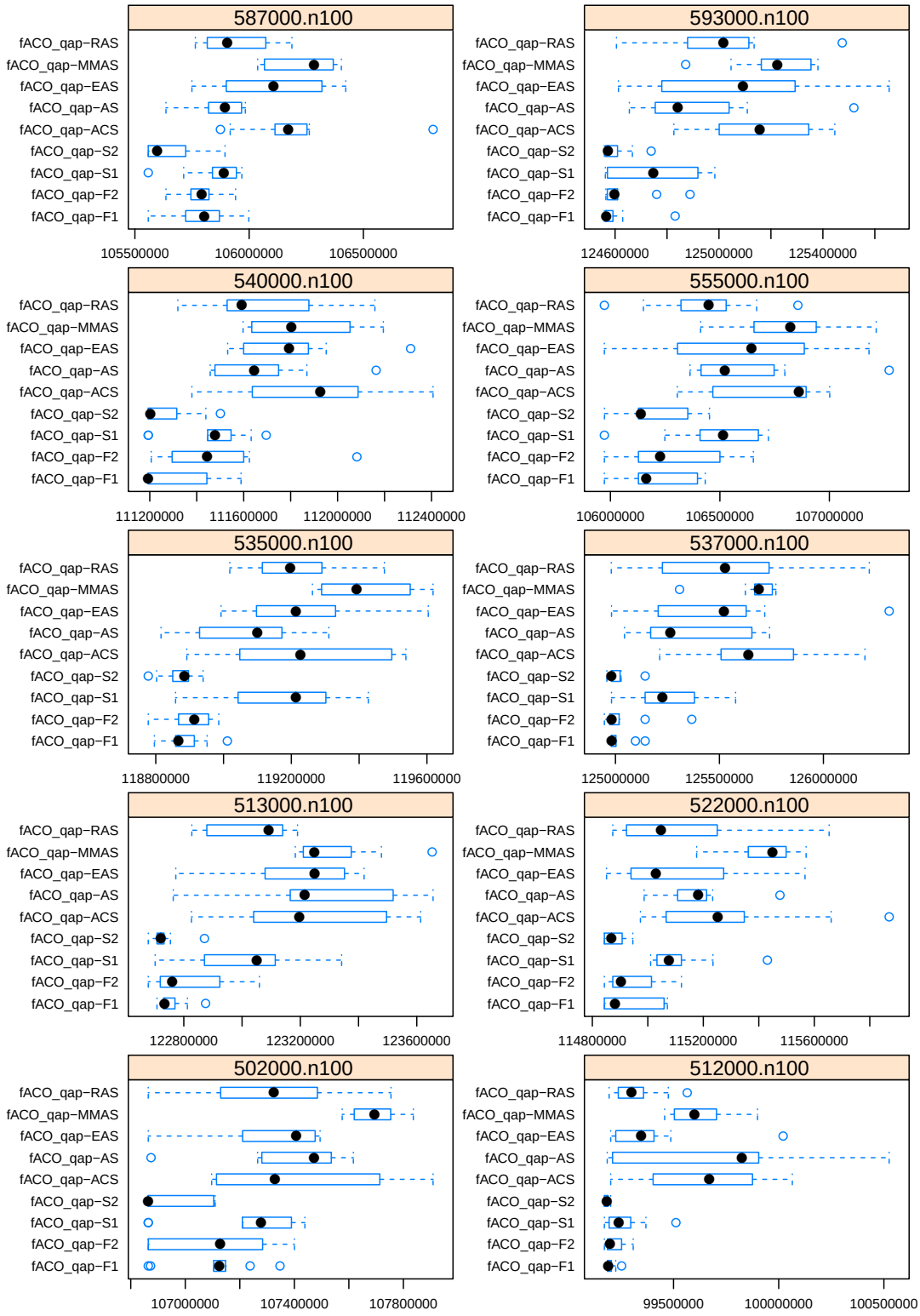

86000.n80
860000.n80
819000.n80
835000.n80
804000.n80
807000.n80
775000.n80
788000.n80
742000.n80
751000.n80
fACO_qap−RAS
fACO_qap−MMAS
fACO_qap−EAS
fACO_qap−AS
fACO_qap−ACS
fACO_qap−S2
fACO_qap−S1
fACO_qap−F2
fACO_qap−F1
77800000 78200000 78600000 79000000
58000000 58200000 58400000 58600000
70000000 70200000 70400000 70600000
67600000 67800000 68000000 68200000
70000000 70500000 71000000
68400000 68600000 68800000 69000000
72200000 72400000 72600000 72800000
79000000 79200000 79400000 79600000 79800000
69400000 69800000 70200000
69600000 69800000 70000000 70200000 70400000

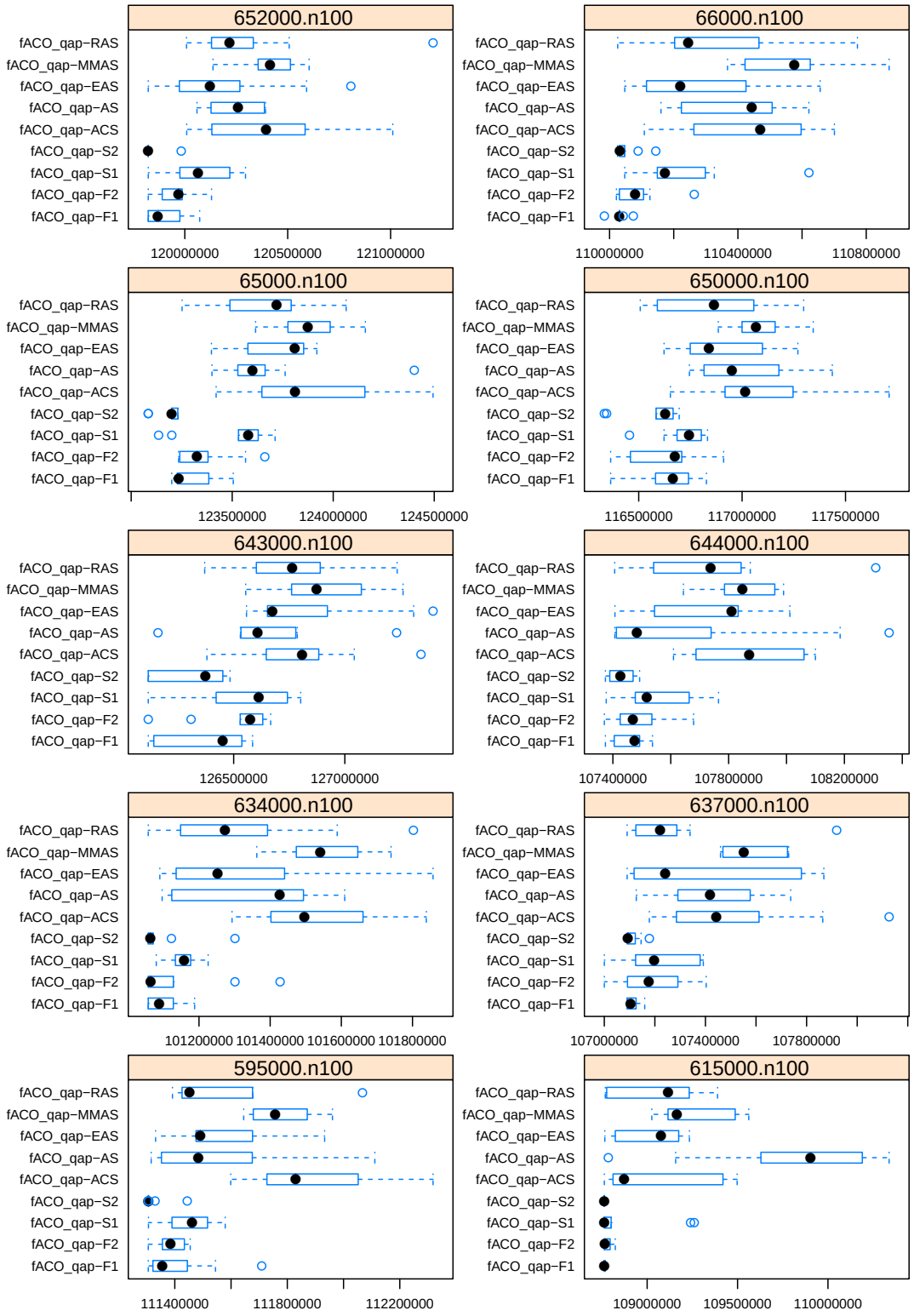

967000.n80
969000.n80
929000.n80
965000.n80
909000.n80
911000.n80
881000.n80
886000.n80
866000.n80
875000.n80
fACO_qap−RAS
fACO_qap−MMAS
fACO_qap−EAS
fACO_qap−AS
fACO_qap−ACS
fACO_qap−S2
fACO_qap−S1
fACO_qap−F2
fACO_qap−F1
63600000 64000000 64400000
67800000 68200000 68600000
66200000 66600000 67000000
77600000 78000000 78400000
67800000 68000000 68200000 68400000
69200000 69600000 70000000 70400000
75600000 76000000 76400000
68200000 68400000 68600000 68800000 69000000
58000000 58200000 58400000 58600000
74600000 74800000 75000000 75200000

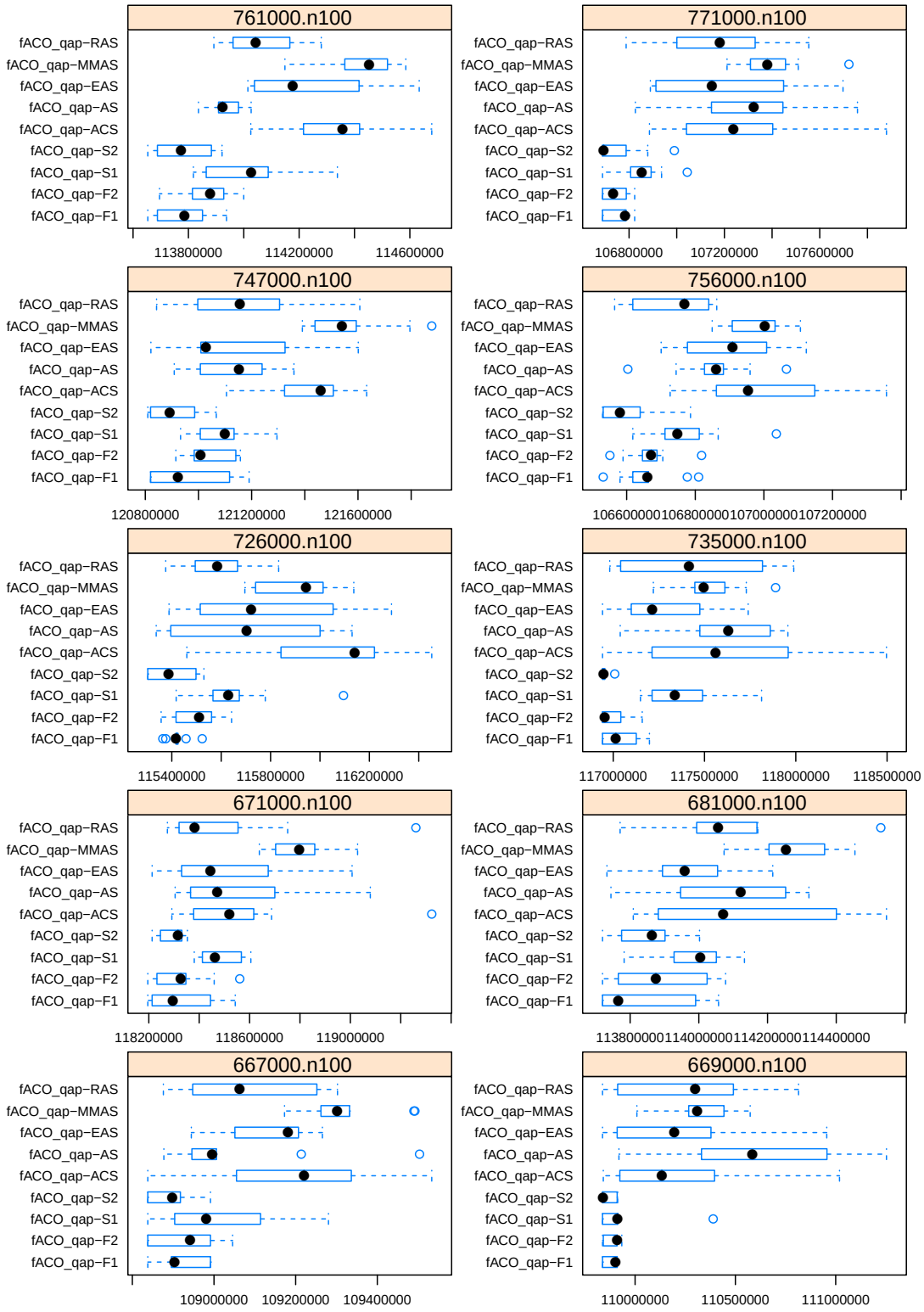

998000.n80
fACO_qap−RAS
fACO_qap−MMAS
fACO_qap−EAS
fACO_qap−AS
fACO_qap−ACS
fACO_qap−S2
fACO_qap−S1
fACO_qap−F2
fACO_qap−F1
72000000
72200000
72400000
978000.n80
71000000
71500000
72000000
72500000
986000.n80
70800000
71000000
71200000
71400000
97000.n80
71000000
71400000
71800000
973000.n80
65400000
65600000
65800000
66000000

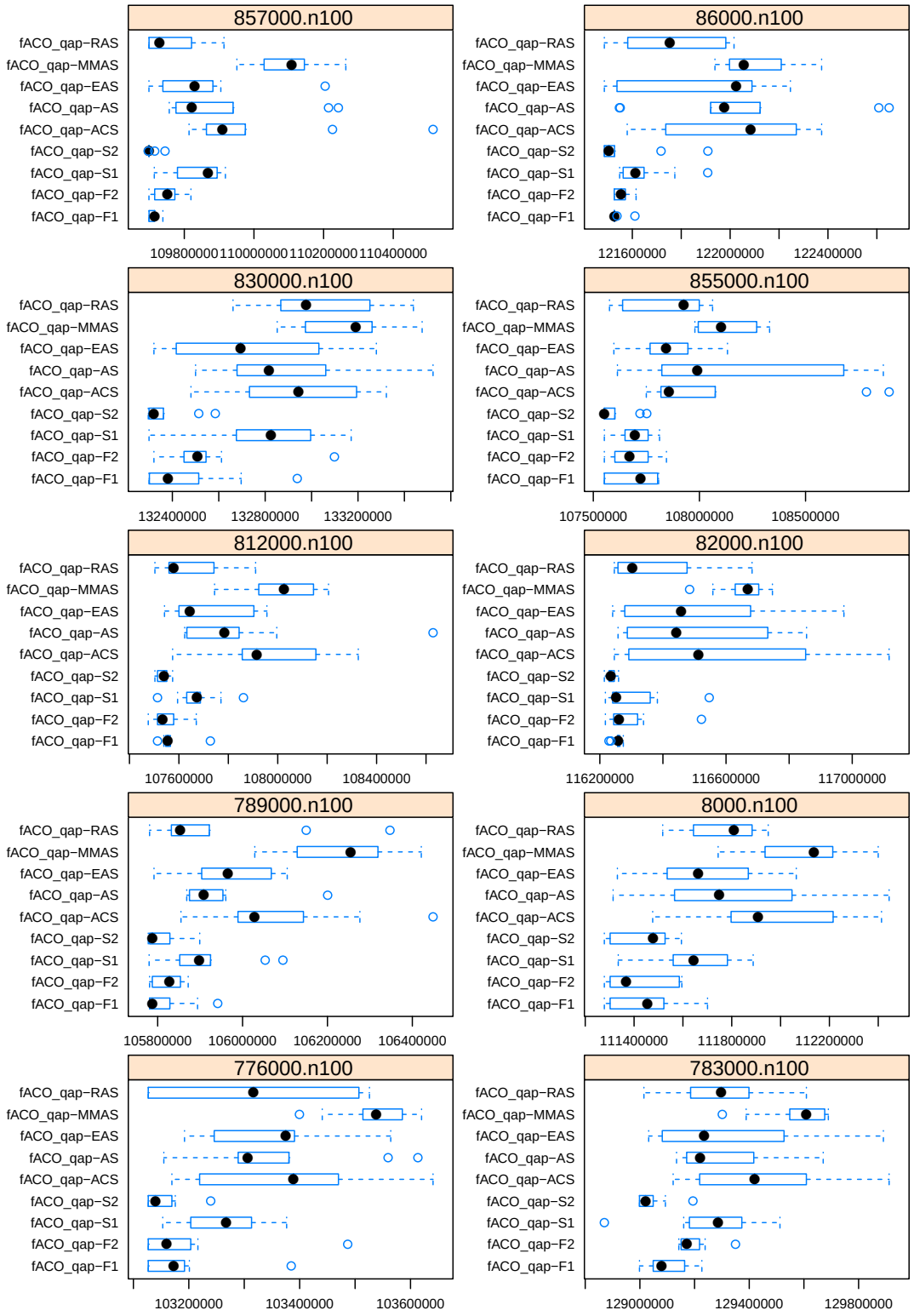

393000.n100
405000.n100
388000.n100
389000.n100
363000.n100
384000.n100
349000.n100
354000.n100
275000.n100
31000.n100
fACO_qap−RAS
fACO_qap−MMAS
fACO_qap−EAS
fACO_qap−AS
fACO_qap−ACS
fACO_qap−S2
fACO_qap−S1
fACO_qap−F2
fACO_qap−F1
116500000 117000000 117500000
96400000 96800000 97200000
117000000 117500000 118000000
114600000 115000000
108200000 108600000 109000000
122000000 122500000 123000000 123500000
105400000 105600000 105800000
120500000 121000000 121500000
104800000 105200000 105600000
119500000 120000000 120500000

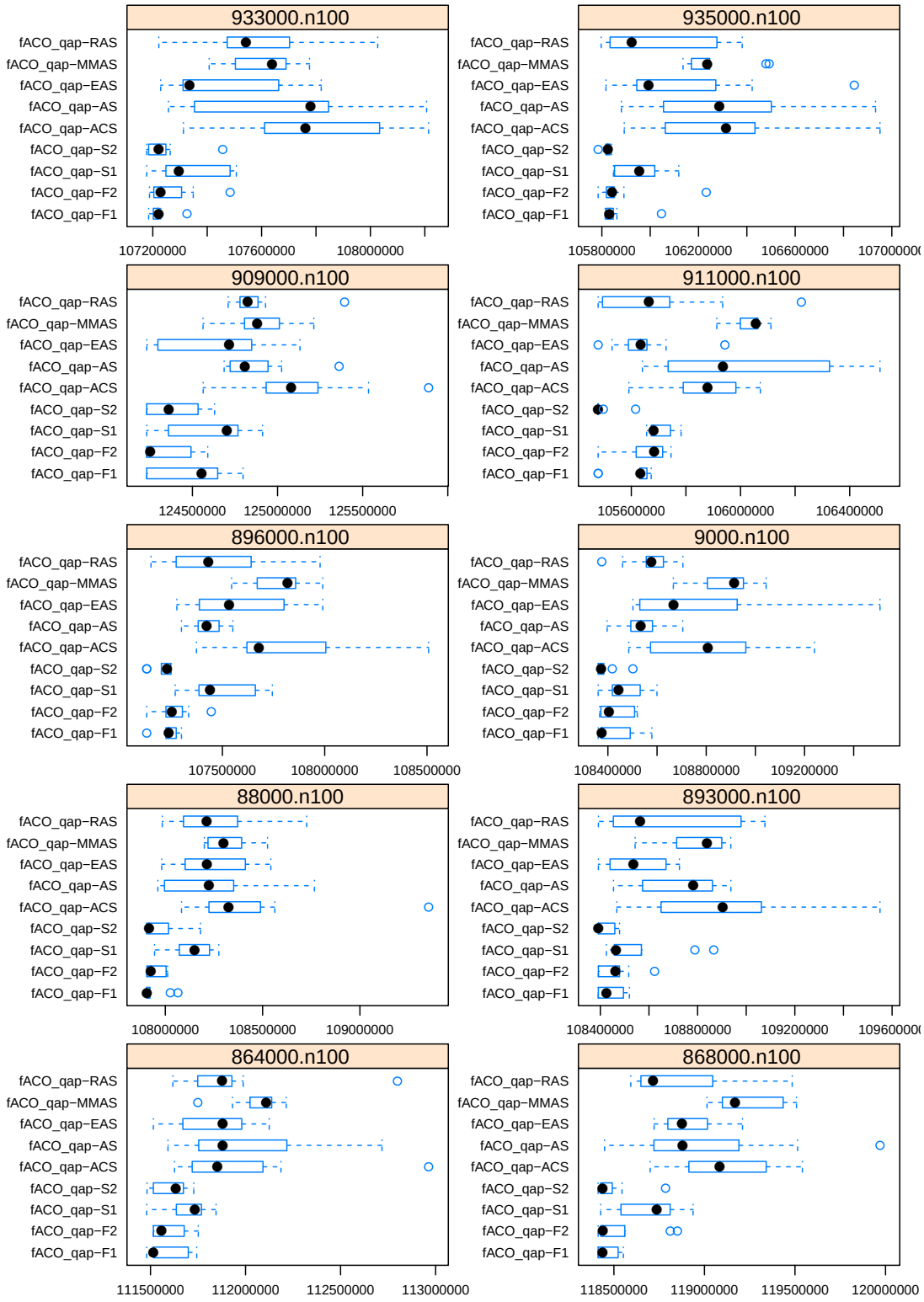

496000.n100
fACO_qap−RAS
fACO_qap−MMAS
fACO_qap−EAS
fACO_qap−AS
fACO_qap−ACS
fACO_qap−S2
fACO_qap−S1
fACO_qap−F2
fACO_qap−F1
108000000
108400000
108800000
5000.n100
107400000
107600000
107800000
108000000
461000.n100
107800000
108000000
108200000
108400000
476000.n100
125000000
125500000
126000000
448000.n100
111800000
112200000
112600000
454000.n100
120500000
121000000
121500000
430000.n100
120000000
120500000
121000000
44000.n100
115000000
115500000
116000000
409000.n100
103200000
103600000
104000000
420000.n100
102800000
103200000
103600000

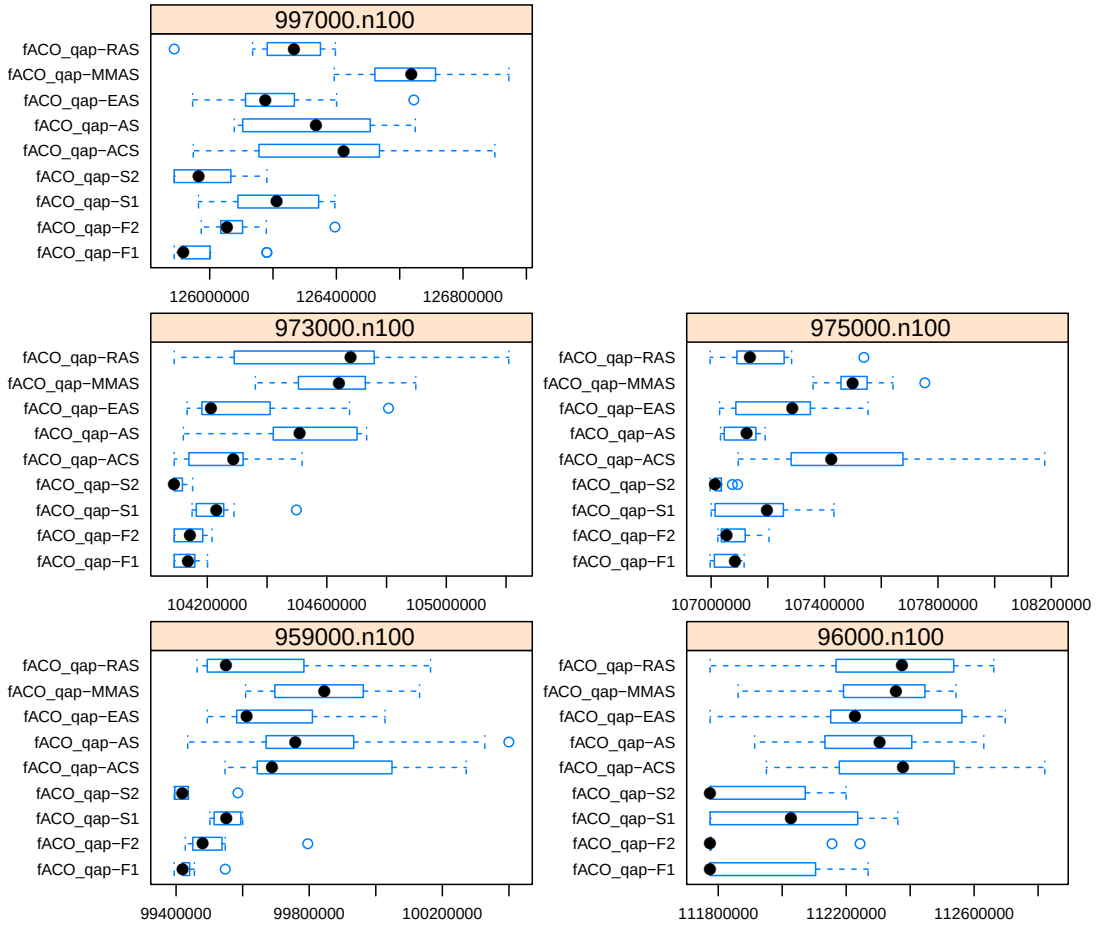

587000.n100
593000.n100
540000.n100
555000.n100
535000.n100
537000.n100
513000.n100
522000.n100
502000.n100
512000.n100
fACO_qap−RAS
fACO_qap−MMAS
fACO_qap−EAS
fACO_qap−AS
fACO_qap−ACS
fACO_qap−S2
fACO_qap−S1
fACO_qap−F2
fACO_qap−F1
105500000
106000000
106500000
124600000
125000000
125400000
111200000
111600000
112000000
112400000
106000000
106500000
107000000
118800000
119200000
119600000
125000000
125500000
126000000
122800000
123200000
123600000
114800000
115200000
115600000
107000000
107400000
107800000
99500000
100000000
100500000


652000.n100
66000.n100
65000.n100
650000.n100
643000.n100
644000.n100
634000.n100
637000.n100
595000.n100
615000.n100
fACO_qap−RAS
fACO_qap−MMAS
fACO_qap−EAS
fACO_qap−AS
fACO_qap−ACS
fACO_qap−S2
fACO_qap−S1
fACO_qap−F2
fACO_qap−F1
120000000 120500000 121000000
110000000 110400000 110800000
123500000 124000000 124500000
116500000 117000000 117500000
126500000 127000000
107400000 107800000 108200000
101200000 101400000 101600000 101800000
107000000 107400000 107800000
111400000 111800000 112200000
109000000 109500000 110000000

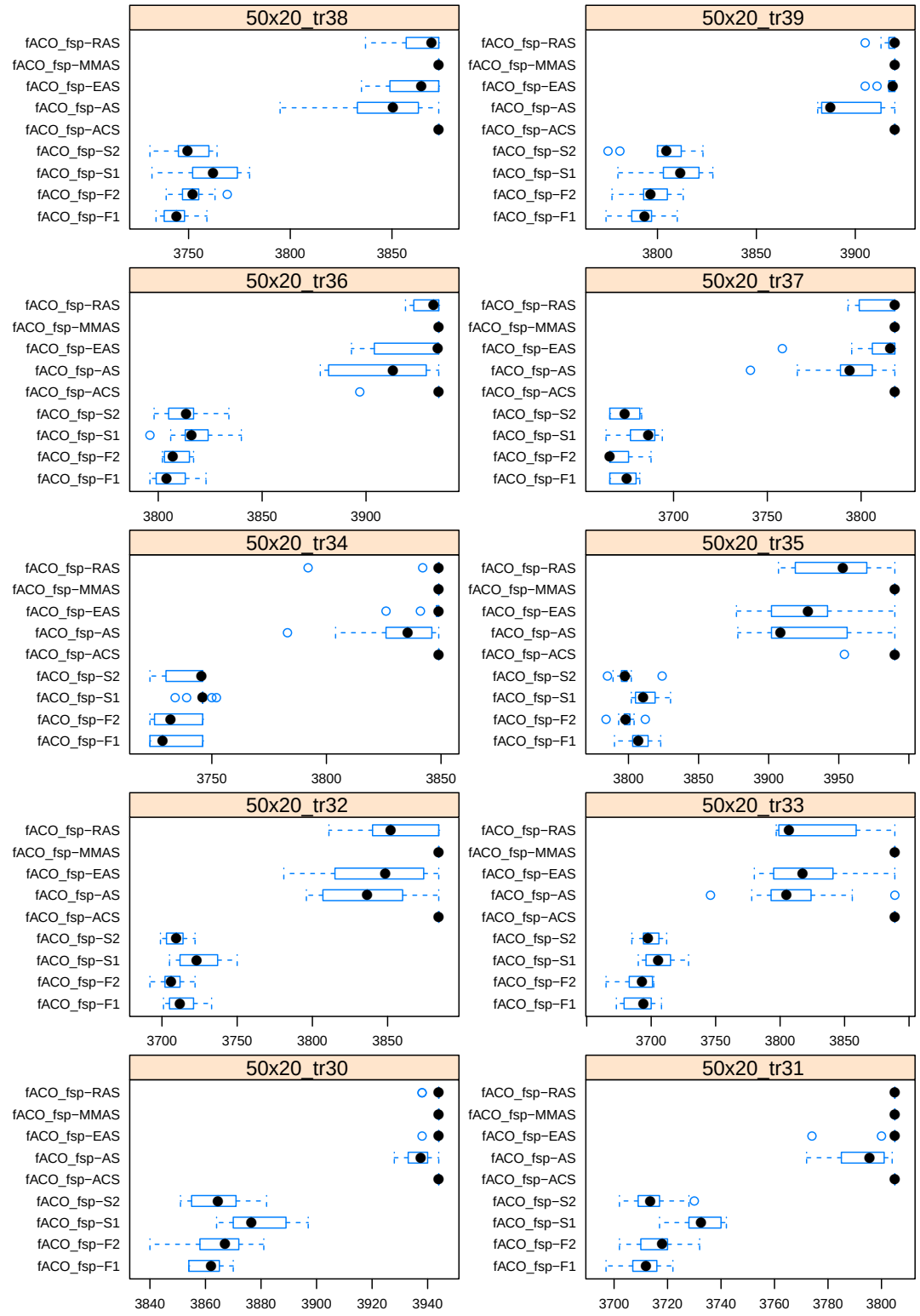

761000.n100
fACO_qap−RAS
fACO_qap−MMAS
fACO_qap−EAS
fACO_qap−AS
fACO_qap−ACS
fACO_qap−S2
fACO_qap−S1
fACO_qap−F2
fACO_qap−F1
113800000
114200000
114600000
771000.n100
106800000
107200000
107600000
747000.n100
120800000
121200000
121600000
756000.n100
106600000106800000107000000107200000
726000.n100
115400000
115800000
116200000
735000.n100
117000000
117500000
118000000
118500000
671000.n100
118200000
118600000
119000000
681000.n100
113800000114000000114200000114400000
667000.n100
109000000
109200000
109400000
669000.n100
110000000
110500000
111000000

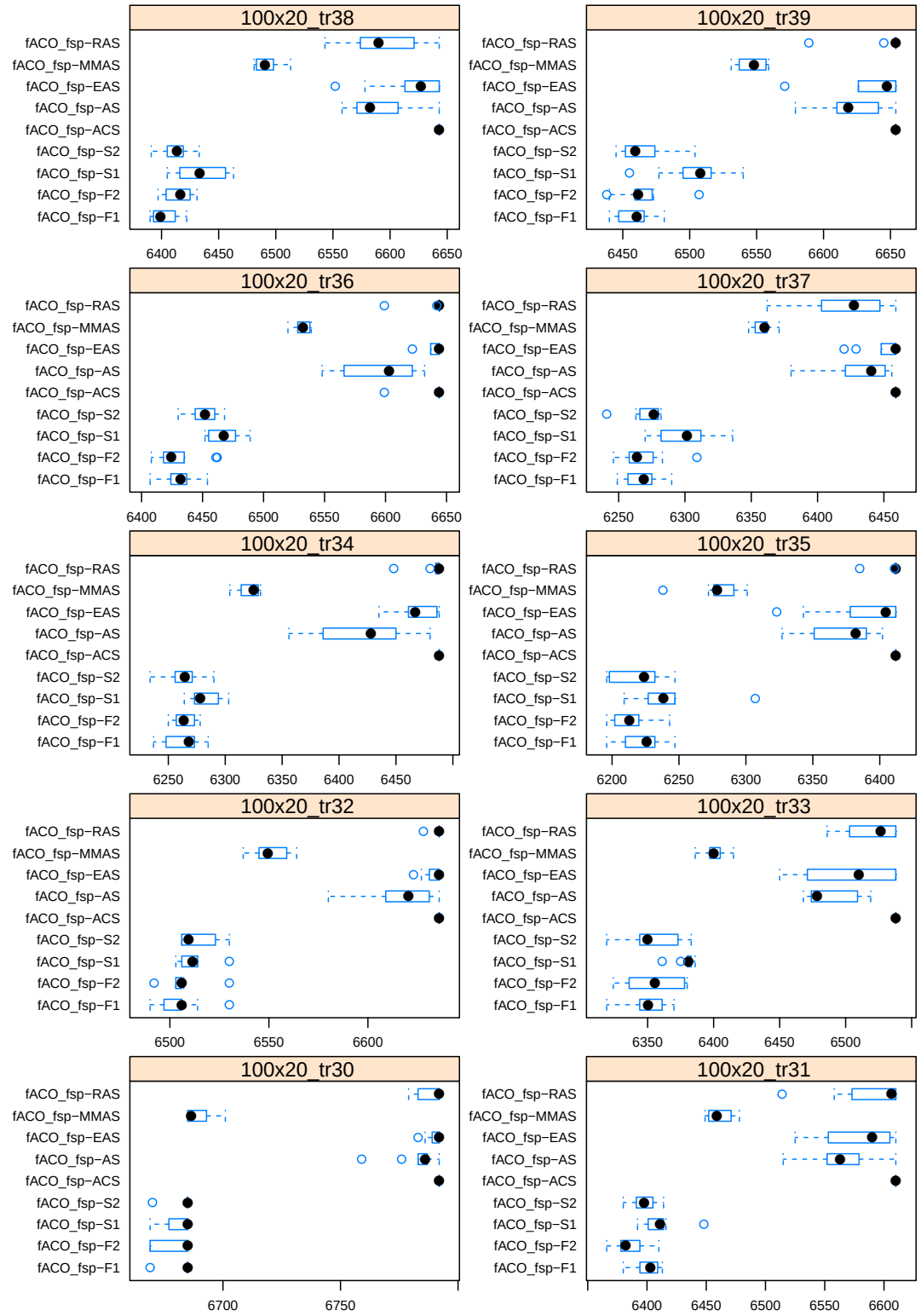

857000.n100
fACO_qap−RAS
fACO_qap−MMAS
fACO_qap−EAS
fACO_qap−AS
fACO_qap−ACS
fACO_qap−S2
fACO_qap−S1
fACO_qap−F2
fACO_qap−F1
109800000 110000000 110200000 110400000
86000.n100
121600000 122000000 122400000
830000.n100
132400000 132800000 133200000
855000.n100
107500000 108000000 108500000
812000.n100
107600000 108000000 108400000
82000.n100
116200000 116600000 117000000
789000.n100
105800000 106000000 106200000 106400000
8000.n100
111400000 111800000 112200000
776000.n100
103200000 103400000 103600000
783000.n100
129000000 129400000 129800000


933000.n100
935000.n100
909000.n100
911000.n100
896000.n100
9000.n100
88000.n100
893000.n100
864000.n100
868000.n100
fACO_qap−RAS
fACO_qap−MMAS
fACO_qap−EAS
fACO_qap−AS
fACO_qap−ACS
fACO_qap−S2
fACO_qap−S1
fACO_qap−F2
fACO_qap−F1
107200000 107600000 108000000
105800000 106200000 106600000
124500000 125000000 125500000
105600000 106000000 106400000
107500000 108000000 108500000
108400000 108800000 109200000
108000000 108500000 109000000
108400000 108800000 109200000
111500000 112000000 112500000 113000000
118500000 119000000 119500000 120000000


997000.n100
fACO_qap−RAS
fACO_qap−MMAS
fACO_qap−EAS
fACO_qap−AS
fACO_qap−ACS
fACO_qap−S2
fACO_qap−S1
fACO_qap−F2
fACO_qap−F1
126000000
126400000
126800000
973000.n100
104200000
104600000
105000000
975000.n100
107000000
107400000
107800000
108200000
959000.n100
99400000
99800000
100200000
96000.n100
111800000
112200000
112600000

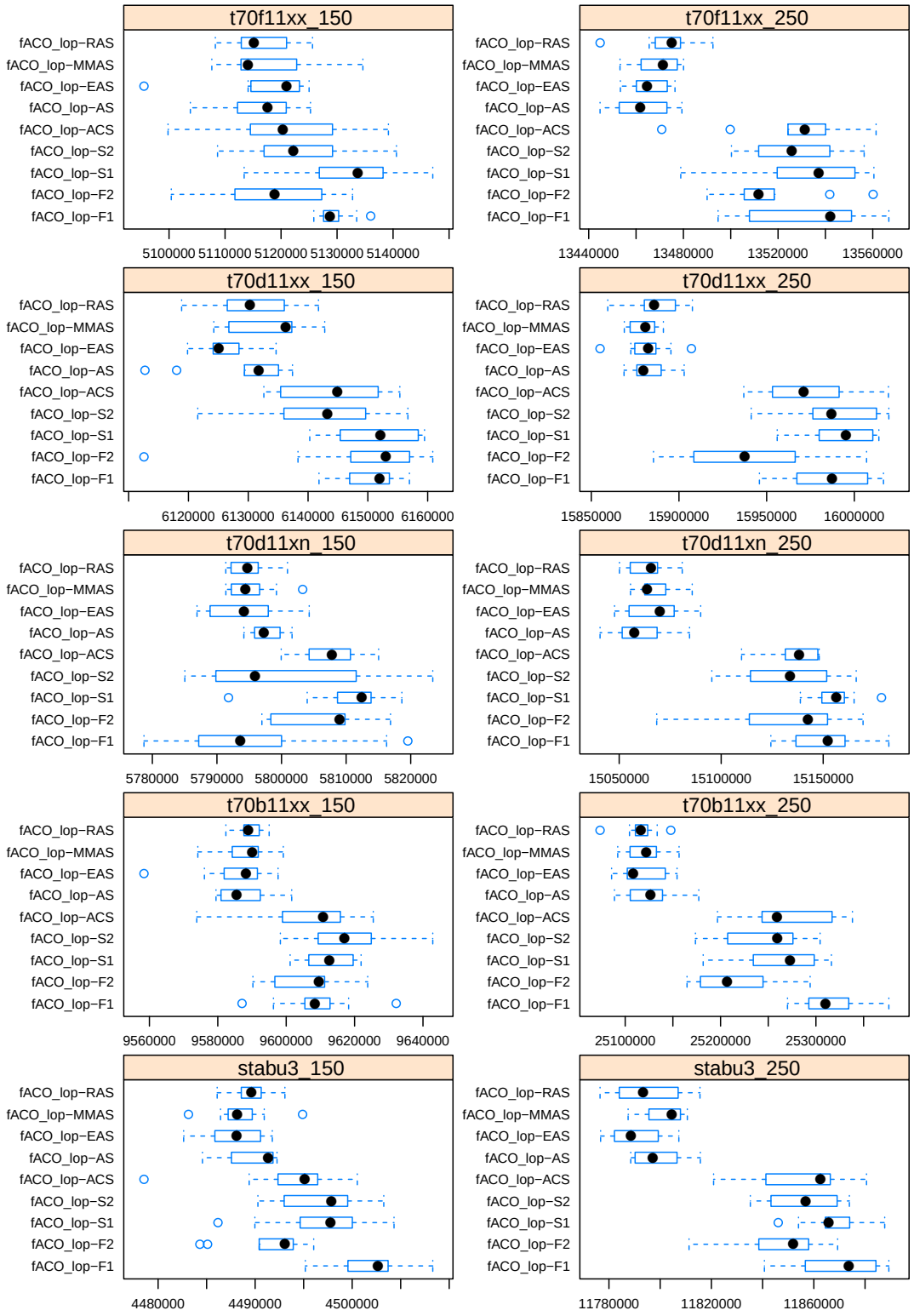
Permutation Flow Shop Problem

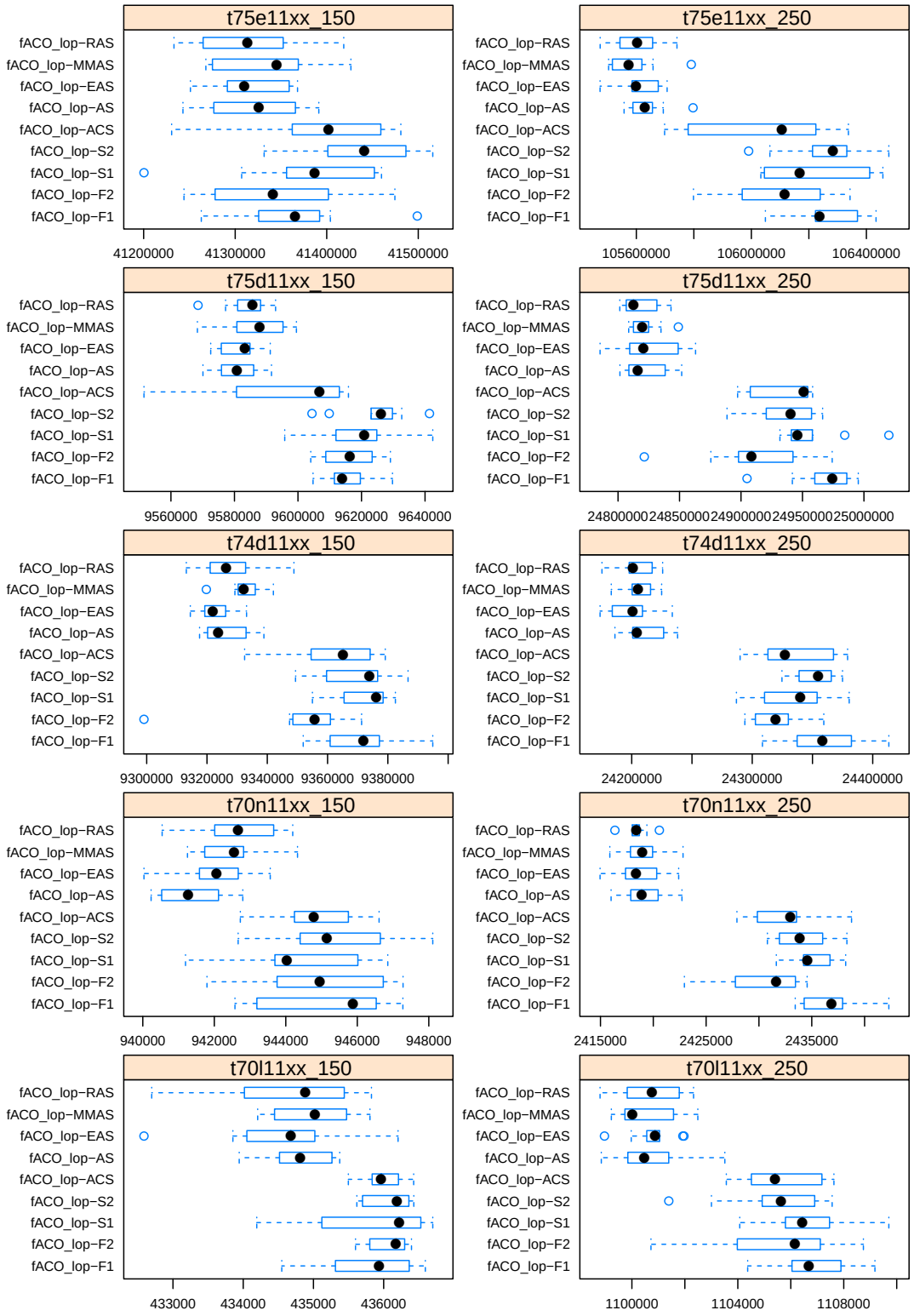

50x20_tr38
50x20_tr39
50x20_tr36
50x20_tr37
50x20_tr34
50x20_tr35
50x20_tr32
50x20_tr33
50x20_tr30
50x20_tr31
fACO_fsp−RAS
fACO_fsp−MMAS
fACO_fsp−EAS
fACO_fsp−AS
fACO_fsp−ACS
fACO_fsp−S2
fACO_fsp−S1
fACO_fsp−F2
fACO_fsp−F1

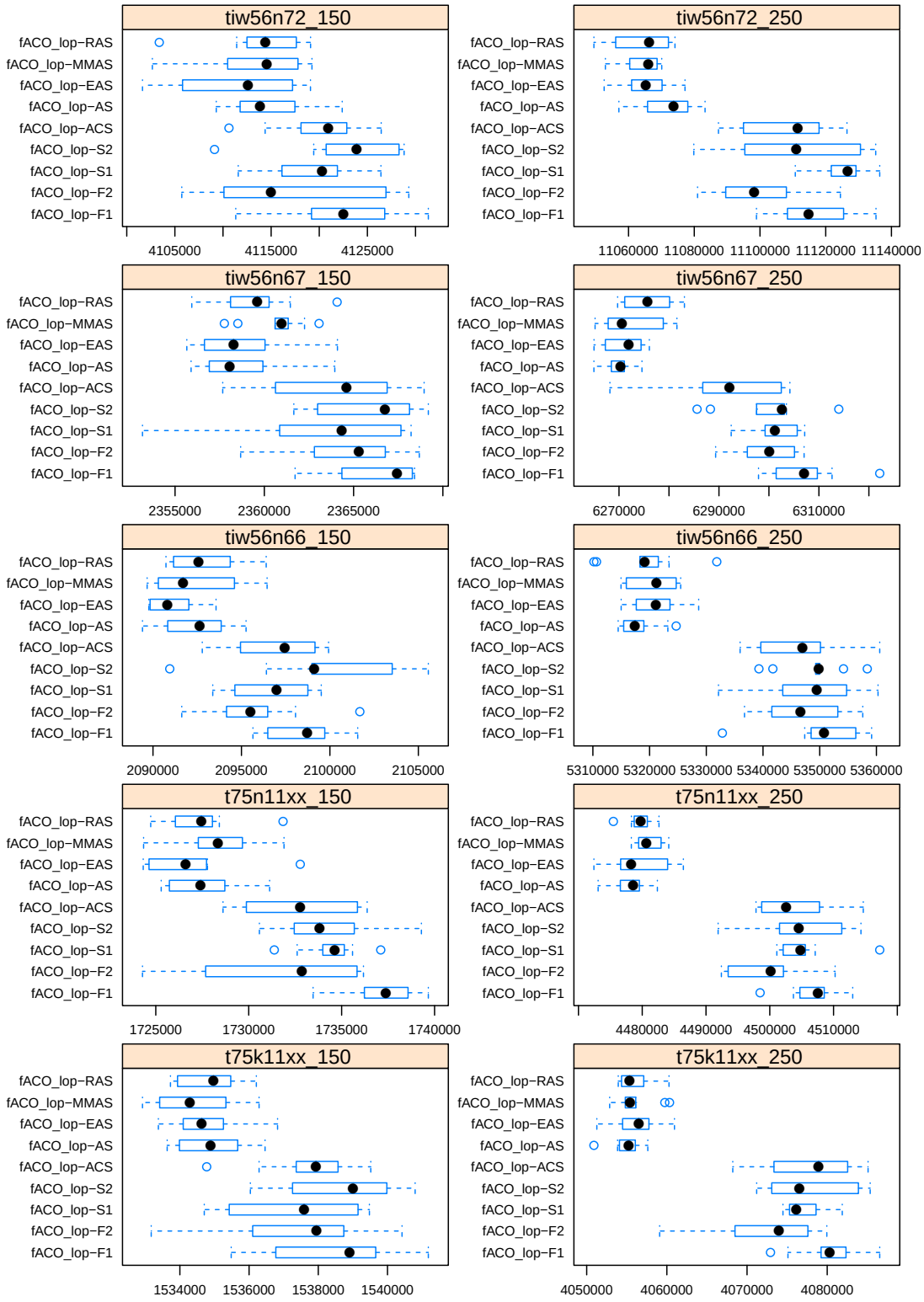

100x20_tr38
100x20_tr39
100x20_tr36
100x20_tr37
100x20_tr34
100x20_tr35
100x20_tr32
100x20_tr33
100x20_tr30
100x20_tr31
fACO_fsp−RAS
fACO_fsp−MMAS
fACO_fsp−EAS
fACO_fsp−AS
fACO_fsp−ACS
fACO_fsp−S2
fACO_fsp−S1
fACO_fsp−F2
fACO_fsp−F1

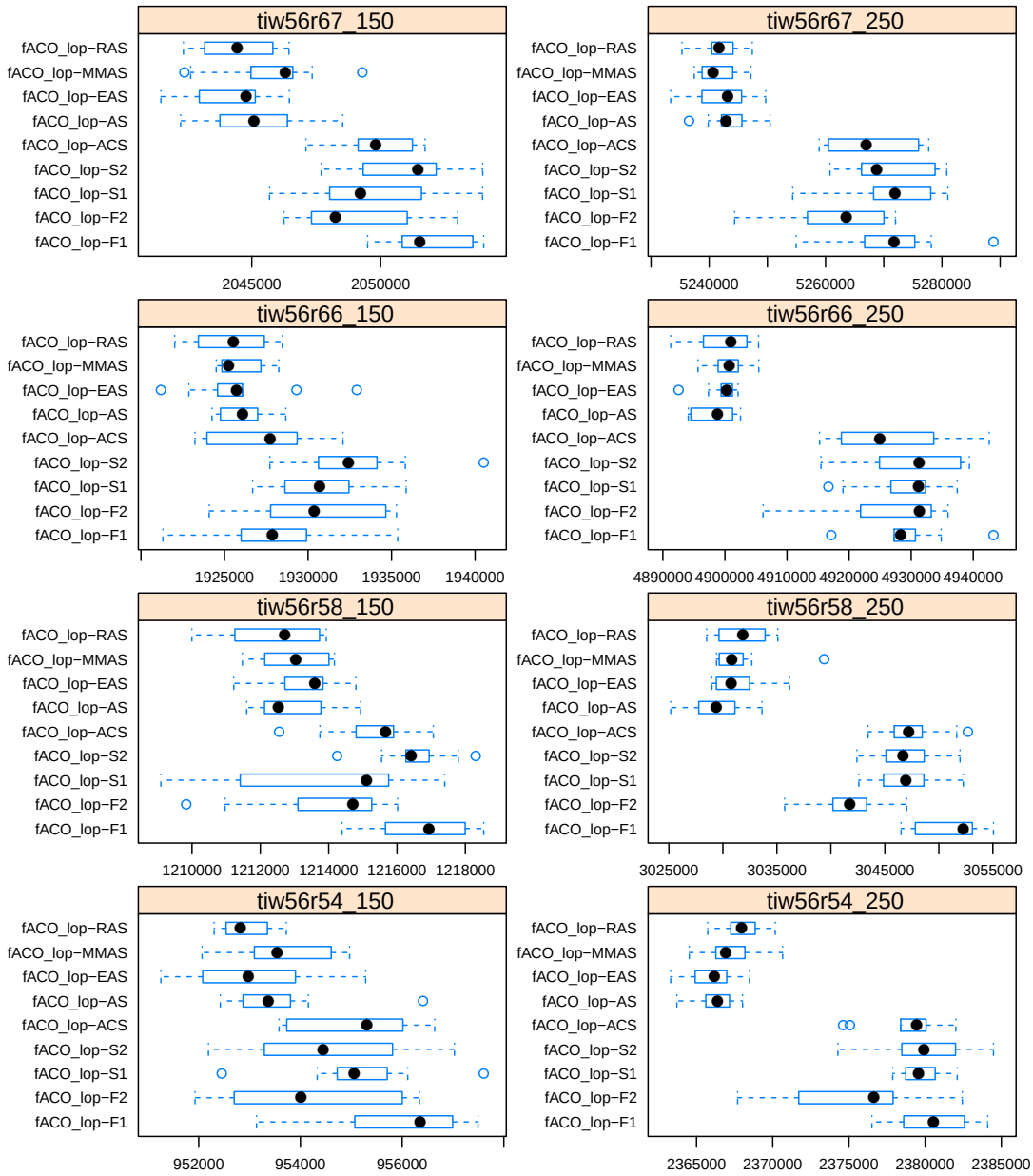
Linear Ordering Problem


stabu2_150
fACO_lop−RAS
fACO_lop−MMAS
fACO_lop−EAS
fACO_lop−AS
fACO_lop−ACS
fACO_lop−S2
fACO_lop−S1
fACO_lop−F2
fACO_lop−F1
4305000 4310000 4315000 4320000 4325000
stabu2_250
11400000 11450000 11500000
be75tot_150
12180000 12220000 12260000
be75tot_250
30650000 30750000 30850000
be75oi_150
2235000 2240000
be75oi_250
5840000 5860000 5880000 5900000
be75np_150
7130000 7140000 7150000 7160000 7170000
be75np_250
17600000 17650000 17700000 17750000
be75eec_150
3460000 3465000 3470000 3475000
be75eec_250
8800000 8820000 8840000 8860000 8880000


t70f11xx_150
fACO_lop−RAS
fACO_lop−MMAS
fACO_lop−EAS
fACO_lop−AS
fACO_lop−ACS
fACO_lop−S2
fACO_lop−S1
fACO_lop−F2
fACO_lop−F1
5100000 5110000 5120000 5130000 5140000
t70f11xx_250
13440000 13480000 13520000 13560000
t70d11xx_150
6120000 6130000 6140000 6150000 6160000
t70d11xx_250
15850000 15900000 15950000 16000000
t70d11xn_150
5780000 5790000 5800000 5810000 5820000
t70d11xn_250
15050000 15100000 15150000
t70b11xx_150
9560000 9580000 9600000 9620000 9640000
t70b11xx_250
25100000 25200000 25300000
stabu3_150
4480000 4490000 4500000
stabu3_250
11780000 11820000 11860000


t75e11xx_150
t75e11xx_250
t75d11xx_150
t75d11xx_250
t74d11xx_150
t74d11xx_250
t70n11xx_150
t70n11xx_250
t70l11xx_150
t70l11xx_250
fACO_lop−RAS
fACO_lop−MMAS
fACO_lop−EAS
fACO_lop−AS
fACO_lop−ACS
fACO_lop−S2
fACO_lop−S1
fACO_lop−F2
fACO_lop−F1
41200000 41300000 41400000 41500000
105600000 106000000 106400000
9560000 9580000 9600000 9620000 9640000
24800000 24850000 24900000 24950000 25000000
9300000 9320000 9340000 9360000 9380000
24200000 24300000 24400000
940000 942000 944000 946000 948000
2415000 2425000 2435000
433000 434000 435000 436000
1100000 1104000 1108000


tiw56n72_150
tiw56n72_250
tiw56n67_150
tiw56n67_250
tiw56n66_150
tiw56n66_250
t75n11xx_150
t75n11xx_250
t75k11xx_150
t75k11xx_250
fACO_lop−RAS
fACO_lop−MMAS
fACO_lop−EAS
fACO_lop−AS
fACO_lop−ACS
fACO_lop−S2
fACO_lop−S1
fACO_lop−F2
fACO_lop−F1
4105000 4115000 4125000
11060000 11080000 11100000 11120000 11140000
2355000 2360000 2365000
6270000 6290000 6310000
2090000 2095000 2100000 2105000
5310000 5320000 5330000 5340000 5350000 5360000
1725000 1730000 1735000 1740000
4480000 4490000 4500000 4510000
1534000 1536000 1538000 1540000
4050000 4060000 4070000 4080000


tiw56r67_150
tiw56r67_250
tiw56r66_150
tiw56r66_250
tiw56r58_150
tiw56r58_250
tiw56r54_150
tiw56r54_250
fACO_lop−RAS
fACO_lop−MMAS
fACO_lop−EAS
fACO_lop−AS
fACO_lop−ACS
fACO_lop−S2
fACO_lop−S1
fACO_lop−F2
fACO_lop−F1
2045000
2050000
5240000
5260000
5280000
1925000
1930000
1935000
1940000
4890000
4900000
4910000
4920000
4930000
4940000
1210000
1212000
1214000
1216000
1218000
3025000
3035000
3045000
3055000
952000
954000
956000
2365000
2370000
2375000
2380000
2385000

Bibliography

[1] Christian Blum and Marco Dorigo. The hyper-cube framework for ant colony optimization. *IEEE Transactions on Systems, Man, and Cybernetics – Part B*, 34(2):1161–1172, 2004.

[2] Christian Blum and M. Sampels. An ant colony optimization algorithm for shop scheduling problems. *Journal of Mathematical Modelling and Algorithms*, 3(3):285–308, 2004.

[3] Jean-Louis Deneubourg, S. Aron, S. Goss, and J.-M. Pasteels. The self-organizing exploratory pattern of the Argentine ant. *Journal of Insect Behavior*, 3(2):159–168, 1990.

[4] Marco Dorigo and Thomas Stützle. *Ant Colony Optimization*. MIT Press, Cambridge, MA, 2004.

[5] B. Esfahbod. P, np, np-hard and np-complete classes. http://en.wikipedia.org/wiki/File:P_np_np-complete_np-hard.svg.

[6] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Thomas Stützle, and Mauro Birattari. The irace package, iterated race for automatic algorithm configuration. Technical Report TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium, 2011.

[7] Manuel López-Ibáñez and Thomas Stützle. The automatic design of multi-objective ant colony optimization algorithms. *IEEE Transactions on Evolutionary Computation*, 16(6):861–875, 2012.

[8] Rafael Martí, Gerhard Reinelt, and Abraham Duarte. A benchmark library and a comparison of heuristic methods for the linear ordering problem. *Computational Optimization and Applications*, 51(3):1297–1317, 2012.

[9] D. Merkle and M. Middendorf. Ant colony optimization with global pheromone evaluation for scheduling a single machine. *Applied Intelligence*, 18(1):105–111, 2003.

[10] C. Rajendran and H. Ziegler. Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs. *European Journal of Operational Research*, 155(2):426–438, 2004.

[11] G. Reinelt. TSPLIB. http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95. Version visited last on 15 June 2012.

[12] Thomas Stützle. $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System for the quadratic assignment problem. Technical Report AIDA–97–4, FG Intellektik, FB Informatik, TU Darmstadt, Germany, July 1997.

[13] Thomas Stützle. An ant approach to the flow shop problem. In *Proceedings of the 6th European Congress on Intelligent Techniques & Soft Computing (EUFIT'98)*, volume 3, pages 1560–1564. Verlag Mainz, Aachen, Germany, 1998.

[14] Thomas Stützle. ACOTSP: A software package of various ant colony optimization algorithms applied to the symmetric traveling salesman problem, 2002.

[15] Thomas Stützle and Holger H. Hoos. $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System. *Future Generation Computer Systems*, 16(8):889–914, 2000.